

Multidimensional Range Search

- Static collection of records.
 - No inserts, deletes, changes.
 - Only queries.
- Each record has k key fields.
- Multidimensional query.
 - Given k ranges $[l_i, u_i]$, $1 \leq i \leq k$.
 - Report all records in collection such that $l_i \leq k_i \leq u_i$, $1 \leq i \leq k$.

Multidimensional Range Search

- All employees whose age is between 30 and 40 and whose salary is between \$40K and \$70K.
- All cities with an annual rainfall between 40 and 60 inches, population between 100K and 200K, average temperature ≥ 70 F, and number of horses between 1025 and 2500.

Data Structures For Range Search

- Unordered sequential list.
- Sorted tables.
 - k tables.
 - Table i is sorted by i 'th key.
- Cells.
- ✓ k -d trees.
- ✓ Range trees.
- k -fold trees.
- k -ranges.

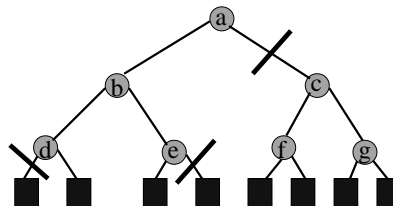
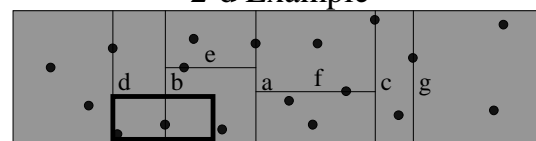
Performance Measures

- $P(n,k)$.
 - Preprocessing time to construct search structure for n records, each has k key fields.
 - For many applications, this time needs only to be reasonable.
- $S(n,k)$.
 - Total space needed by the search structure.
- $Q(n,k)$.
 - Time needed to answer a query.

k -d Tree

- Binary tree.
- At each node of tree, pick a key field to partition records in that subtree into two approximately equal groups.
 - Pick field i with max spread in values.
 - Select median key value, m .
- Node stores i and m .
- Records with $k_i \leq m$ in left subtree.
- Records with $k_i > m$ in right subtree.
- Stop when partition size ≤ 8 or 16 (say).

2-d Example



Performance

- $P(n,k) = O(n \log n)$.
 - $O(\log n)$ levels.
 - $O(n)$ time to find all medians at each level of the tree.

Performance

- $S(n,k) = O(n)$.
 - $O(n)$ needed for the n records.
 - Tree takes $O(n)$ space.

Performance

- $Q(n,k)$ depends on shape of query.
 - $O(n^{1-1/k} + s)$, where s is number of records that satisfy the query. Bound on worst-case query time.
 - $O(\log n + s)$, average time when query is almost cubical and a small fraction of the n records satisfy the query.
 - $O(s)$, average time when query is almost cubical and a large fraction of the n records satisfy the query.

Range Trees— $k=1$

- Sorted array on single key.

10	12	15	20	24	26	27	29	35	40	50	55
----	----	----	----	----	----	----	----	----	----	----	----

- $P(n,1) = O(n \log n)$.
- $S(n,1) = O(n)$.
- $Q(n,1) = O(\log n + s)$.

Range Trees— $k=2$

- Let the two key fields be x and y .
- Binary search tree on x .
- x value used in a node is the median x value for all records in that subtree.
- Records with x value \leq median are in left subtree.
- Records with larger x value in right subtree.

Range Trees— $k=2$

- Each node has a sorted array on y of all records in the subtree.
 - Root has sorted array of all n records.
 - Left and right subtrees, each have a sorted array of about $n/2$ records.
- Stop partitioning when # records in a partition is small enough (say 8).

Example

- a-g are x values.
- x-range of a node begins at min x value in subtree and ends at max x value in subtree.

Example—Search

- If x-range of root is contained in x-range of query, search SA for records that satisfy y-range of query. Done.
- If entire x-range of query $\leq x$ ($> x$) value in root, recursively search left (right) subtree.

Example—Search

- If x-range of query contains value in root, recursively search left and right subtrees.

Performance

- $P(n,2) = O(n \log n)$.
 - $O(n \log n)$ – sort all records by y for the SAs.
 - $O(n)$ time to find all medians at each level of the tree.

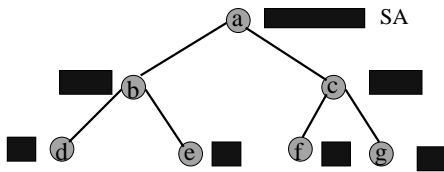
Performance

- $P(n,2) = O(n \log n)$.
 - $O(n)$ time to construct SAs at each level of the tree from SAs at preceding level.
 - $O(\log n)$ levels.

Performance

- $S(n,2) = O(n \log n)$.
 - $O(n)$ needed for the SAs and nodes at each level.
 - $O(\log n)$ levels.

Performance



- $Q(n,2) = O(\log^2 n + s)$.
 - At each level of the binary search tree, at most 2 SAs are searched.
 - $O(\log n)$ levels.

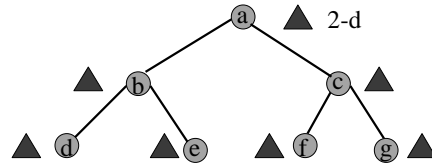
Range Trees— $k=3$

- Let the three key fields be w , x and y .
- Binary search tree on w .
- w value used in a node is the median w value for all records in that subtree.
- Records with w value \leq median in left subtree.
- Records with larger w value in right subtree.

Range Trees— $k=3$

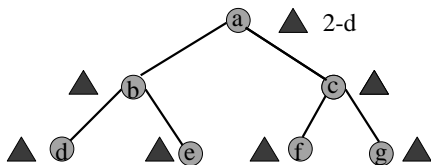
- Each node has a 2-d range tree on x and y of all records in the subtree.
- Stop partitioning when # records in a partition is small enough (say 8).

Example



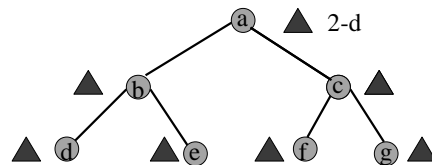
- a - g are w values.
- w -range of a node begins at min w value in subtree and ends at max w value in subtree.

Example—Search



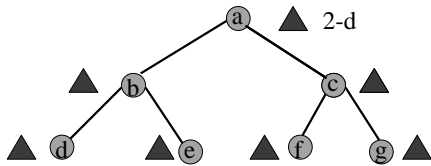
- If w -range of root is contained in w -range of query, search 2-d range tree in root for records that satisfy x - and y -ranges of query. Done.
- If entire w -range of query $\leq w$ ($> w$) value in root, recursively search left (right) subtree.

Example—Search



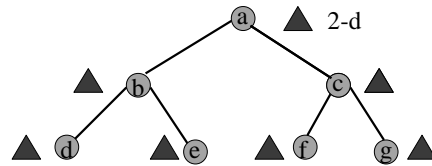
- If w -range of query contains value in root, recursively search left and right subtrees.

Performance — 3-d Range Tree



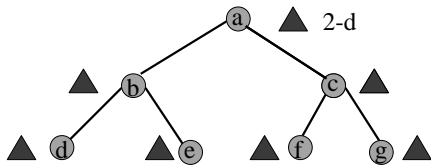
- $P(n,3) = O(n \log^2 n)$.
 - $O(n)$ time to find all medians at each level of the tree.

Performance — 3-d Range Tree



- $P(n,3) = O(n \log^2 n)$.
 - $O(n \log n)$ time to construct 2-d range trees at each level of the tree from SAs at preceding level.
 - $O(\log n)$ levels.

Performance — 3-d Range Tree



- $S(n,3) = O(n \log^2 n)$.
 - $O(n \log n)$ needed for the 2-d range trees and nodes at each level.
 - $O(\log n)$ levels.

Performance — 3-d Range Tree

- $Q(n,3) = O(\log^3 n + s)$.
 - At each level of the binary search tree, at most 2 2-d range trees are searched.
 - $O(\log^2 n + s_i)$ time to search each 2-d range tree. s_i is # records in the searched 2-d range tree that satisfy query.
 - $O(\log n)$ levels.

Performance—k-d Range Tree

- $P(n,k) = O(n \log^{k-1} n)$, $k > 1$.
- $S(n,k) = O(n \log^{k-1} n)$.
- $Q(n,k) = O(\log^k n + s)$.