DS for Range Reporting

- 1. Computation Model
- 2. R-Tree
  - 2.1 Structure
  - 2.2 Updates
- 3. B-Tree
  - 3.1 History
  - 3.1 Structure and operations
- 4. Kd-Tree
  - 4.1 Structure
  - 4.2 Operations
- 5. O-Tree
  - 5.1 Structure

> Yufei Tao. Practical and Theoretical I/O-Efficient Data Structures for Range Reporting. VLDB'15 Summer School Lecture Notes, 2015.

- range reporting query. Let P be a set of N points in $\mathbb{R}^2$. Given an axis-parallel rectangle $q = [x_1, x_2] \times [y_1, y_2]$, a range reporting query reports all the points of P that are covered by q.
  - application: "find all the restaurants in the area";
  - relational database application: TaxPayer(id, sal, age), select id from TaxPayer where 10<=sal<=20 and 50<=age<=60;
- theme: IO-efficient data structure for Range Reporting

1. Computation Model

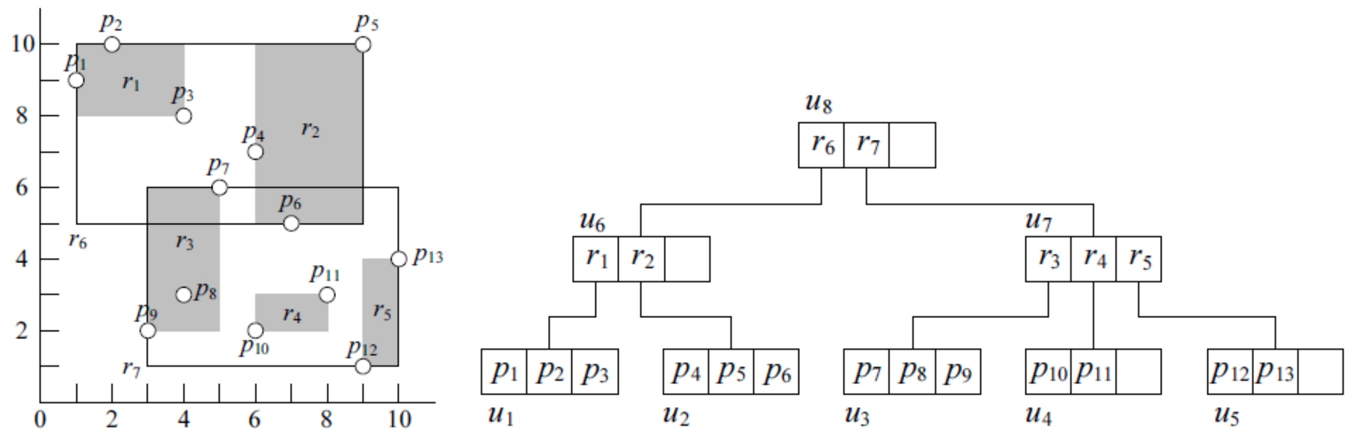
- RAM model. An $O(NlogN)$ algorithm means the algorithm is able to solve the problem by performing $O(NlogN)$ "basic operations"
  - standard CUP work or access a memory location
- External memory model. Since an I/O is rather expensive (1-10 milliseconds)
  - space: M words of memory and unbounded disk with block size being B words
  - space complexity: number of disk blocks occupied
  - time complexity: number of I/Os (read B words from disk to memory or write B words conversely), i.e., CPU calculation and memory accesses are free
  - $O(N/B)$ instead of $O(N)$ linear cost

2. R-Tree

> N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In SIGMOD, pages 322-331, 1990.

- properties:
  - query cost: $O(N/B)$ (efficient in practice but no performance guarantee)
  - insertion cost: $O(B \log B \log_B N)$
  - deletion cost: query + $B \log_B N$ insertion

2.1 Structure



(a) Data and MBRs (b) The structure

Figure 1: An R-tree

- Minimal bounding rectangle (MBR): smallest rectangle that tightly encloses all data points, see r_1, r_2, ... in the figure for example
  - good MBR: square-like by reducing perimeter

- leaf node: $b/4$–b points where $b = \Theta(B)$
- non-leaf node: $b/4$–b children, and store an MBR for each child
- $O(N/B)$ space and of $O(\log_B N)$ height

2.2 Updates

Goal: create square-like MBRs

- Insertions
  - strategy: find a leaf to insert the point, recursively split if overflow (i.e., b+1 points/children)
  - choosing a subtree to split: the one with minimal increase of perimeter, and find the one having the smallest area if ties exist
  - node split: use an axis-orthogonal cut and find a better one

- Deletions
  - strategy: find the leaf node containing p by a range query using p itself and delete p
  - underflow: the number of points/children < b/4, by remove the node and re-insert all points/MBRs

3. B-Tree

3.1 History

- Binary search tree
  - Basic operations: SEARCH, MINIMUM, MAXIMUM, PREDECESSOR, SUCCESSOR, INSERT, and DELETE

in $O(h)$ time
    - Structure: key[x.lchild] <= key[x] <= key[x.rchild]
    - random BST: the expectation height of an n-element randomly constructed BST is $O(\lg n)$

- Red-Black tree
    - a variant of BST with performance guarantee $O(\lg n)$
    - height of RB tree: $2\lg(n+1)$

- B-Tree
    - Balanced BST in disk: IO-efficient, and the size of a node = page size (xKB), i.e., a node can have t children (t~50-2000)
    - height: $h \le \log_t \frac{n+1}{2}$

3.1 Structure and operations

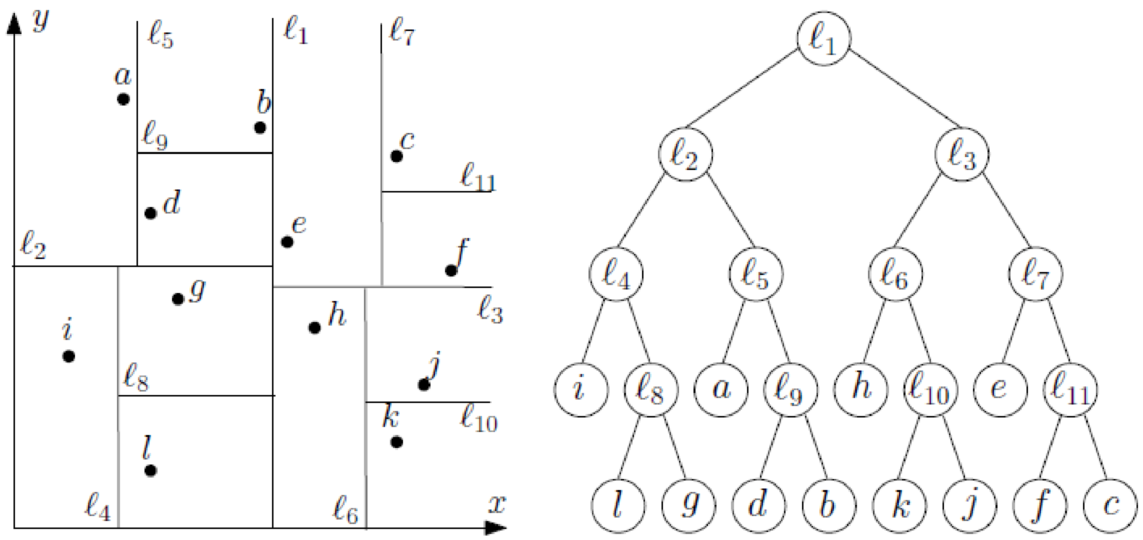
- internal node branches p and leaf node size b, usually $b = B$ and $p = B^c$ for some constant $c \in (0, 1]$
- height $O(\log_p(N/b))$
- re-balancing operations:
    - split: half-half split u to u, u' if $|u| = b$, and make both u' a new child of $u.parent$;
    - merge: merge u and its sibling u' together;
    - both take $O((b+p)/B)$ I/Os

- Update:
    - insert: (1) find a leaf node to insert; and (b) recursively split if needed;
    - delete: (1) find a leaf node to delete; and (b) recursively merge if needed;
    - both take $O(b/B) + (p/B)\log_p(N/b)$ I/Os

4. Kd-Tree

- answer a range reporting quety with optimal cost

4.1 Structure



- a binary tree
- non-leaf node: a splitting line dividing the points into 2 half-half set, note that splitting dimension

from root the any leaf follows either x-y-x-… or y-x-y-…

- leaf node: at most B points in $\mathbb{R}^2$, and hence at least $B/2$
- k-selection algorithm: one-pass alg for splitting
- height $O(\log(N/B))$, construction I/O $O((N/b)\log(N/B))$, space $O(N/B)$

4.2 Operations

- Query: each non-leaf node corresponding to a bounding rectangle as shown in the figure
    - cost: $O(\sqrt{(N/B)}+K/B)$ I/Os
- insert and delete

5. O-Tree

- the same space and query complexities as kd-tree, i.e., $O(N/B)$ and $O(\sqrt{(N/B)}+K/B)$, but has the advantage of being updateable in $O(\log_B N)$ amortized I/Os per insertion and deletion

5.1 Structure