CS144 Notes: Spatial Index

Main question

- Many queries are location-sensitive, especially queries from mobile devices.
  - e.g., where is a nearby gas station?
- Common types of spatial queries
  - *Range queries*: Given a “range” (say, a rectangular bounding box), return the set of points within the range
  - *Nearest-neighbor queries*: Return the closest point to a given point
  - *Where-am-I queries*: Given a point, which object overlaps with the point?
  - *Partial-match queries*: Given a value for a specific dimension, return all points matching the value in that dimension
    - E.g. What countries are on equator?
- Q: How can we support spatial queries efficiently?

Spatial queries in SQL

Table Point(x, y):

- Q1: SQL query for “find all points in the rectangle 10 < x < 20 and 30 < y < 40”?

- Q2: SQL query for “find the nearest point to (10, 20)”?

- Q: How do we answer Q1? Any efficient way? B+tree?
  - Reminder
    - Data is stored in disks
    - Data is stored in the unit of “blocks” in disk

- Example:
  1,000,000 points within of 0 < x < 100 and 0 < y < 100. Uniformly distributed.
  100 points per disk block
    - Q: How to build indexes?
    - Q: We may cluster points based on the primary (clustered) index. What about the secondary index?
    - Q: How will the points be stored in disk blocks?
    - Q: How many points are retrieved?
    - Q: Is it helpful to have a secondary index for the given query?

- Q: How do we answer Q2? Any efficient way?
    - Q: Pick a range in each dimension. Ask the range query. Select the closest point to the target within the range. Any problem? Two possible outcome
        - Q: Case 1. No point within the selected range. What should we do?
            - Q: How should we pick the initial range?

        - Q: Case 2. A point found in the range. Is it the nearest point?

            - Q: When do we have to worry about a closer point outside the range?

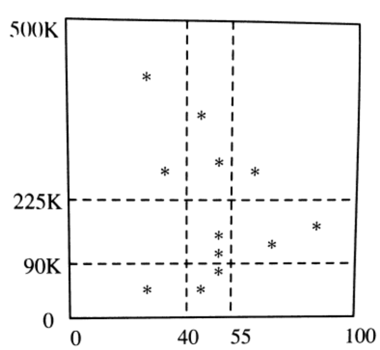
    - Q: 1,000,000 points uniformly distributed between 0 < x < 1,000 and 0 < y < 1,000. Nearest point to (100, 200). What is a good choice for the initial range d of 100 – d < x < 100 + d and 200 – d < y < 200 + d?

- Q: How can we build an index for "multidimensional" or "spatial" queries?

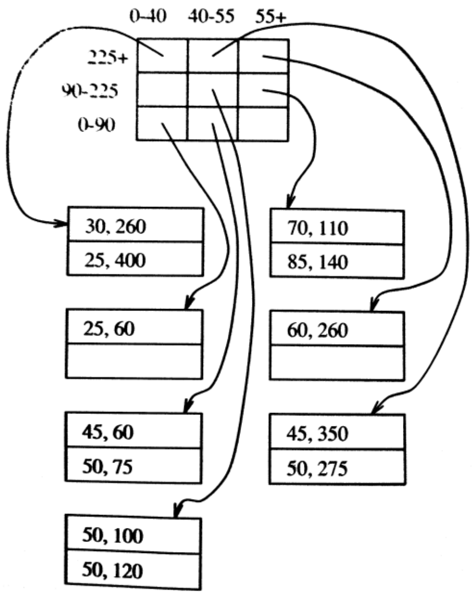
Grid File

- Space is partitioned into a "grid"
    - In each dimension, "gridlines" partition the space into "stripes".
    - The number of gridlines and their spacing may vary among dimensions
- Example: (age, salary) data
    - (25, 60K), (45, 60K), (50, 75K), (50, 100K), (50, 120K), (70, 110K), (85, 140K), (30, 260K), (25, 400K), (45, 350K), (50, 275K), (60, 260K)

    - Conceptual grid file: (2 data points per bucket)



- Grid file data structure.
    - (n x n) array of bucket pointers with their range values
    - A set of data buckets storing the actual data points



        - E.g., Lower left bucket stores all points in (0≤ x<40, 0≤ y<90)

- Querying grid file
    - Partial-match queries
        - Q: Find all customers with a salary of $200K? What buckets need to be searched?
    - Range queries
        - Q: Find all customers aged 35-45 and with a salary of 50K-100K? What buckets need to be searched?
    - Nearest-neighbor queries
        - Q: Find the point nearest to (45, 200K)? What buckets need to be searched?

- Updating grid file
    - Q: Insert (52, 199K). What should we do when the bucket is full?

        - Choice 1: Add overflow bucket. Pros and cons?

        - Choice 2: Reorganize by adding or moving the grid lines.
            - Challenge: adding a grid line splits all buckets along the line
            - #1: Split with a vertical line? Where?

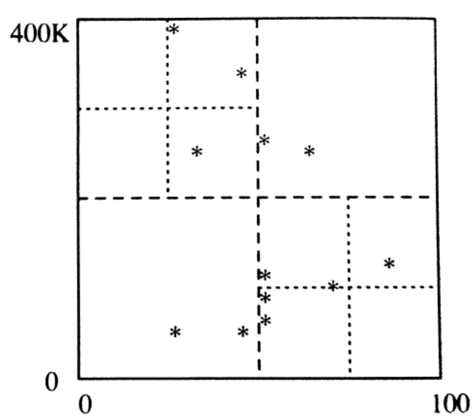
            - #2: Split with a horizontal line? Where?

    - Generally, grid file is not suitable for a dynamic dataset with many insertions

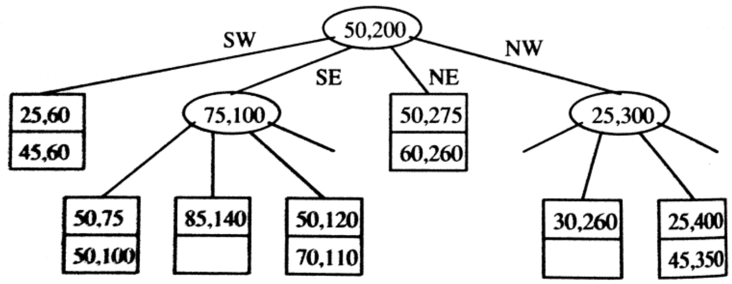
Quad Tree

- A region is recursively partitioned into four equal-sized quadrants until all points in a region can fit into a single disk bucket.

    - Conceptual quad tree example: (each bucket can store 2 points)



- Quad tree data structure:



    - Each internal node (circle) splits a region into four quadrants and stores pointers to its children
    - Leaf nodes corresponds to one disk bucket and stores data points (rectangle)

- Querying a quad tree
  - Partial-match queries
    - Q: Find all customers with a salary of $200K? What buckets need to be searched?
  - Range queries
    - Q: Find all customers aged 35-45 and with a salary of 50K-100K? What buckets need to be searched?
  - Nearest-neighbor queries
    - Q: Find the point nearest to (45, 200K)? What buckets need to be searched?
  - At each internal node, check overlap of the query region with each quadrant and search any quadrant with overlap
- Updating a quad tree
  - Q: Insert (52, 199K). What should we do when the bucket is full?

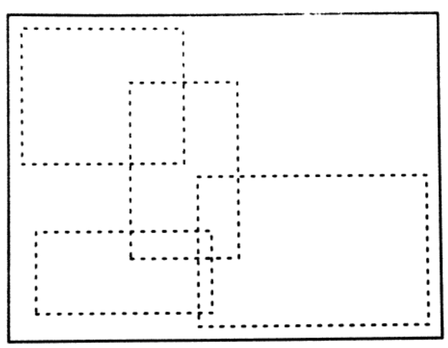
- Q: Is the tree always balanced? Is the tree "well occupied"?

    - Comments: Quad tree is simple to implement yet its rigidity may lead to skews (and low performance) in the data structure

- Note:
  - A region does not need to be split into (2 x 2) quadrants. Many systems split a region into (n x n) quadrants, where n is a configuration parameter.
  - It is possible to extend the quad-tree algorithms to insert "shapes" not just "points" into the tree. We do not cover this extension in this class.
  - Oracle and Microsoft SQL Server support quad tree.

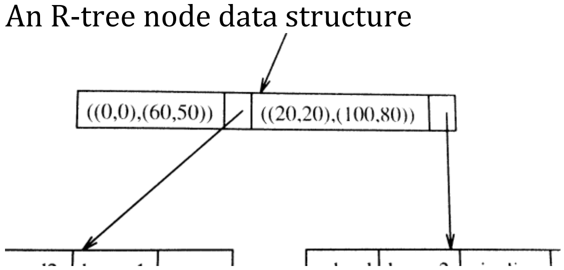
R-tree (Region tree)

- Each internal node of an R-tree contains a set of (arbitrary-sized) subregions as its children. Data points are stored at leaf nodes.
  - Example: An R-tree node region and the subregions of its children



    - Note: When there is an overlap in subregions, a data point may be stored in any one of the overlapping subregions.

  - An R-tree node data structure



- Querying an R-tree
  - Q: Find all customers aged 35-45 and with a salary of 50K-100K? What buckets need to be searched?

- At each internal node, check overlap of the query region with each subregion and search *all* subregions with overlap

- Updating an R-tree
  - Insert a new point.
    - Q: What if there is no subregion containing the point?

      - In subregion expansion, minimal expansion is often preferred.

    - Q: What if there is no space in the corresponding subregion?

      - In subregion split, we generally want the two subregions to be as small as possible while covering all data points in the region.

- Note:
  - It is possible to extend the R-tree algorithms to insert “shapes” not just “points” into the tree. We do not cover this extension in this class.
  - Oracle and MySQL support R-tree.