Multidimensional Data Structures
Multidimensional Data Structures

- An important source of media data is geographic data.

- A geographic information system (GIS) stores information about some physical region of the world.

- A map is just viewed as a 2-dimensional image, and certain “points” on the map are considered to be of interest.

- These points are then stored in one of many specialized data structures.
  - $k$-d Trees
  - Point Quadtrees
  - MX-Quadtrees

- Alternatively, we may wish to store certain rectangular regions of the map.

- We will study one data structure – the R-tree – that is used to store such rectangular data.
Example Maps

(a) Map with Marked Points       (b) Map with Marked Regions
$k$-D Trees

- Used to store $k$ dimensional point data.
- It is *not* used to store region data.
- A 2-d tree (i.e., for $k = 2$) stores 2-dimensional point data while a 3-d tree stores 3-dimensional point data, and so on.
Node Structure

\[
\text{nodetype} = \text{record} \\
\text{INFO: infotype;} \\
\text{XVAL: real;} \\
\text{YVAL: real;} \\
\text{LLINK: } \uparrow\text{nodetype} \\
\text{RLINK: } \uparrow\text{nodetype} \\
\text{end}
\]

<table>
<thead>
<tr>
<th>INFO</th>
<th>XVAL</th>
<th>YVAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- INFO field is any user-defined type whatsoever.
- XVAL and YVAL denote the coordinates of a point associated with the node.
- LLINK and RLINK fields point to two children.
2-d trees, formally

Level of nodes is defined in the usual way (with root at level 0).

**Def:** A 2-d tree is any binary tree satisfying the following condition:

1. If $N$ is a node in the tree such that $\text{level}(N)$ is even, then every node $M$ in the subtree rooted at $N.LLINK$ has the property that $M.XVAL < N.XVAL$ and every node $P$ in the subtree rooted at $N.RLINK$ has the property that $P.XVAL \geq N.XVAL$.

2. If $N$ is a node in the tree such that $\text{level}(N)$ is odd, then every node $M$ in the subtree rooted at $N.LLINK$ has the property that $M.YVAL < N.YVAL$ and every node $P$ in the subtree rooted at $N.RLINK$ has the property that $P.YVAL \geq N.YVAL$. 
Example 2-d Trees

(a) BanjaLuka (19,45)
(b) BanjaLuka (19,45)
(c) BanjaLuka (19,45)
(d) Toslic (38,38)
(e) Sinj (4,4)

Level:0
Level:1
Level:2
Level:3
Insertion/Search in 2-d Trees

To insert a node $N$ into the tree pointed to by $T$, do as follows:

- Check to see if $N$ and $T$ agree on their XVAL and YVAL fields.
- If so, just overwrite node $T$ and we are done.
- Else, branch left if $N.XVAL < T.XVAL$ and branch right otherwise.
- Suppose $P$ denotes the child we are examining. If $N$ and $P$ agree on their XVAL and YVAL fields, just overwrite node $P$ and we are done, else branch left if $N.YVAL < P.YVAL$ and branch right otherwise.
- Repeat this procedure, branching on XVAL’s when we are at even levels in the tree, and on YVALs when we are at odd levels in the tree.
Example of Insertion

Suppose we wish to insert the following points.

<table>
<thead>
<tr>
<th>City</th>
<th>(XVAL,YVAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banja Luka</td>
<td>(19,45)</td>
</tr>
<tr>
<td>Derventa</td>
<td>(40,50)</td>
</tr>
<tr>
<td>Toslic</td>
<td>(38,38)</td>
</tr>
<tr>
<td>Tuzla</td>
<td>(54,35)</td>
</tr>
<tr>
<td>Sinj</td>
<td>(4,4)</td>
</tr>
</tbody>
</table>
Example of Insertion

(a) Splitting of region by Banja Luka  (b) Splitting of region by Derventa

(c) Splitting of region by Toslic  (d) Splitting of region by Sinj
Deletion in 2-d Trees

Suppose $T$ is a 2-d tree, and $(x, y)$ refers to a point that we wish to delete from the tree.

- Search for the node $N$ in $T$ that has $N.XVAL = x$ and $N.YVAL = y$.

- If $N$ is a leaf node, then set the appropriate field (LLINK or RLINK) of $N$’s parent to NIL and return $N$ to available storage.

- Otherwise, either the subtree rooted at $N.LLINK$ (which we will denote by $T_\ell$) or the subtree rooted at $N.RLINK$ (which we will denote by $T_r$) is non-empty.

**Step 1** Find a “candidate replacement” node $R$ that occurs either in $T_i$ for $i \in \{\ell, r\}$.

**Step 2** Replace all of $N$’s non-link fields by those of $R$.

**Step 3** Recursively delete $R$ from $T_i$.

- The above recursion is guaranteed to terminate as $T_i$ for $i \in \{\ell, r\}$ has strictly smaller height than the original tree $T$. 
Finding Candidate Replacement Nodes for Deletion

- The desired replacement node $R$ must bear the same spatial relation to all nodes $P$ in both $T_l$ and $T_r$ that $N$ bore to $P$.
- I.e. if $P$ is to the southwest of $N$, then $P$ must be to the southwest of $R$. If $P$ is to the northwest of $N$, then $P$ must be to the northwest of $R$, and so on.

This means that the desired replacement node $R$ must satisfy the property that:

1. Every node $M$ in $T_l$ is such that: $M.XVAL < R.XVAL$ if $\text{level}(N)$ is even and $M.YVAL < R.YVAL$ if $\text{level}(N)$ is odd.

2. Every node $M$ in $T_r$ is such that: $M.XVAL \geq R.XVAL$ if $\text{level}(N)$ is even and $M.YVAL \geq R.YVAL$ if $\text{level}(N)$ is odd.

- If $T_r$ is not empty, and $\text{level}(N)$ is even, then any node in $T_r$ that has the smallest possible XVAL field in $T_r$ is a candidate replacement node.
- But if $T_r$ is empty, then we might not be able to find a candidate replacement node from $T_l$ (why?).
• In this case, find the node $R'$ in $T_l$ with the smallest possible XVAL field. Replace $N$ with this.

• Set $N.RLINK = N.LLINK$ and set $N.LLINK = NIL$.

• Recursively delete $R'$.
Range Queries in 2-d Trees

- A range query with respect to a 2-d tree $T$ is a query that specifies a point $(x_c, y_c)$, and a distance $r$.
- The answer to such a query is the set of all points $(x, y)$ in the tree $T$ such that $(x, y)$ lies within distance $d$ of $(x_c, y_c)$.
- I.e. A range query defines a circle of radius $r$ centered at location $(x_c, y_c)$, and expects to find all points in the 2-d tree that lie within the circle.
- Recall that each node $N$ in a 2-d tree implicitly represents a region $R_N$.
- If the circle specified in a query has no intersection with $R_N$, then there is no point searching the subtree rooted at node $N$. 
Example Range Query
Point Quadtrees

- Point quadtrees always split regions into *four* parts.

- In a 2-d tree, node $N$ splits a region into two by drawing *one* line through the point $(N.XVAL, N.YVAL)$.

- In a point quadtree, node $N$ splits the region it represents by drawing *both* and horizontal *and* a vertical line through the point $(N.XVAL, N.YVAL)$.

- These four parts are called the NW (northwest), SW (southwest), NE (northeast) and SE (southeast) quadrants determined by node $N$.

- Each of these quadrants corresponds to a child of node $N$. Thus, quadtree nodes may have up to 4 children each.

- Node structure in a point quadtree:

  ```
  qtnodetype = record
      INFO: infotype;
      XVAL: real;
      YVAL: real;
      NW,SW,NE,SE: qtnodetype
  end
  ```
Nodes in Point Quadtrees Implicitly Represent Regions
### Insertion into Point Quadtrees

<table>
<thead>
<tr>
<th>City</th>
<th>(XVAL, YVAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banja Luka</td>
<td>(19, 45)</td>
</tr>
<tr>
<td>Derventa</td>
<td>(40, 50)</td>
</tr>
<tr>
<td>Toslic</td>
<td>(38, 38)</td>
</tr>
<tr>
<td>Tuzla</td>
<td>(54, 35)</td>
</tr>
<tr>
<td>Sinj</td>
<td>(4, 4)</td>
</tr>
</tbody>
</table>
Insertion into Point Quadtrees
(a) Splitting of region by Banja Luka  
(b) Splitting of region by Derventa

(c) Splitting of region by Toslic  
(d) Splitting of region by Tuzla
Insertion into Point Quadtrees

(a) Banja Luka (19,45)
(b) Banja Luka (19,45)
(c) Banja Luka (19,45)
(d) Derventa (40,50)
(e) Toslic (38,38)

These slides may not be duplicated without explicit written permission from Morgan Kaufmann Press.

Principles of Multimedia Database Systems Morgan Kaufmann Copyright ©1997
Deletion in Point Quadtrees

- If the node being deleted is a leaf node, deletion is completely trivial: we just set the appropriate link field of node $N$’s parent to NIL and return the node to available storage.

- As in the case of deletion in 2-d trees, we need to find an appropriate replacement node for non-leaf nodes being deleted.

- Is this easy?

- No. Why? Return to Previous slide.
Expand Node Type

- Expand the node structure `qtnodetype` to a new node structure `newqtnodetype`

- `qtnodetype = record`
  - `INFO: infotype;`
  - `XVAL,YVAL: real;`
  - `XLB,YLB,XUB,YUB: real ∪ {−∞, +∞}`
  - `NW,SW,NE,SE: †qtnodetype`

- When inserting a node $N$ into the tree $T$, we need to ensure that:
  - If $N$ is the root of tree $T$, then $N.XLB = −∞$, $N.YLB = −∞$, $N.XUB = +∞$, $N.YUB = +∞$.
  - If $P$ is the parent of $N$ then the following table describes what $N$’s XLB, YLB, XUB, YUB fields should be, depending upon whether $N$ is the NW, SW, NE, SE child of $P$. We use the notation $w = (P.XUB − P.XLB)$ and $h = (P.YUB − Y.LLB)$.

<table>
<thead>
<tr>
<th>Case</th>
<th>N.XLB</th>
<th>N.XUB</th>
<th>N.YLB</th>
<th>N.YUB</th>
</tr>
</thead>
<tbody>
<tr>
<td>N=P.NW</td>
<td>P.XLB</td>
<td>P.XLB + $w \times 0.5$</td>
<td>P.YLB + $h \times 0.5$</td>
<td>P.YUB</td>
</tr>
<tr>
<td>N=P.SW</td>
<td>P.XLB</td>
<td>P.XLB + $w \times 0.5$</td>
<td>P.YLB</td>
<td>P.YLB + $h \times 0.5$</td>
</tr>
<tr>
<td>N=P.NE</td>
<td>P.XLB + $w \times 0.5$</td>
<td>P.XUB</td>
<td>P.YLB + $h \times 0.5$</td>
<td>P.YUB</td>
</tr>
<tr>
<td>N=P.SE</td>
<td>P.XLB + $w \times 0.5$</td>
<td>P.XUB</td>
<td>P.YLB</td>
<td>P.YLB + $h \times 0.5$</td>
</tr>
</tbody>
</table>
Deletion in Point Quadtrees, Continued

- When deleting an interior node \( N \), we must find a replacement node \( R \) in one of the subtrees of \( N \) (i.e. in one of \( N.W, N.SW, N.NE, N.SE \)) such that:
  * every other node \( R_1 \) in \( N.NW \) is to the north west of \( R \),
  * every other node \( R_2 \) in \( N.SW \) is to the south west of \( R \),
  * every other node \( R_3 \) in \( N.NE \) is to the north east of \( R \) and
  * every other node \( R_4 \) in \( N.SE \) is to the south east of \( R \).

- Consider the figure on the next page.

- Suppose we wish to delete Banja Luka from this quadtree. In this case, one such replacement node can in fact be found, viz. Toslic.

- However, in general, it may not always be possible to find such a replacement node. See the figure in the page after next.
Deletion of Banja Luka
(a) Splitting of region by Banja Luka
(b) Splitting of region by Dervent
(c) Splitting of region by Toslic
(d) Splitting of region by Tuzla
Thus, in general, deletion of an interior node $N$ may require reinsertion of all nodes in the subtrees pointed to by $N.NE$, $N.SE$, $N.NW$ and $N.SW$. In the worst case, this may require almost all nodes to be reinserted.
Range Searches in Point Quadtrees

- Each node in a point quadtree represents a region.
- Do not search regions that do not intersect the circle defined by the query.

\begin{verbatim}
proc RangeQueryPointQuadtree(T: newqtnodetype, C: circle);
   \begin{enumerate}
      \item If region(T) \cap C = \emptyset then Halt
      \item else
         \begin{enumerate}
            \item If (T.XVAL, T.YVAL) \in C then print (T.XVAL, T.YVAL);
            \item RangeQueryPointQuadtree(T.NW, C);
            \item RangeQueryPointQuadtree(T.SW, C);
            \item RangeQueryPointQuadtree(T.NE, C);
            \item RangeQueryPointQuadtree(T.SE, C);
         \end{enumerate}
   \end{enumerate}
end proc
\end{verbatim}
The MX-Quadtree

- For both 2-d trees as well as point quadtrees, the “shape” of the tree depends upon the order in which objects are inserted into the tree.

- In addition, both 2-d trees and point quadtrees split regions into 2 (for 2-trees) or 4 (for point quadtrees) sub-regions – however, the split may be uneven depending upon exactly where the point \((N.XVAL, N.YVAL)\) is located inside the region represented by node \(N\).

- MX-quadtrees attempt to ensure that the shape (and height) of the tree are independent of the number of nodes present in the tree, as well as the order of insertion of these nodes.

- MX-quadtrees also attempt to provide efficient deletion and search algorithms.
The MX-Quadtree

- Assume that the map being represented is “split up” into a grid of size $(2^k \times 2^k)$ for some $k$.
- The application developer is free to choose $k$ as s/he likes to reflect the desired granularity, but once s/he chooses $k$, s/he is required to keep it fixed.

Example:
The MX-Quadtree

- **Node Structure:** Exactly the same as for point quadtrees, except that the root of an MX-quadtree represents the region specified by \( \text{XLB} = 0, \text{XUB} = 2^k, \text{YLB} = 0, \text{YUB} = 2^k \).

- When a region gets “split”, it gets split down the middle. Thus, if \( N \) is a node, then the regions represented by the four children of \( N \) are described by the following table.

<table>
<thead>
<tr>
<th>Child</th>
<th>XLB</th>
<th>XUB</th>
<th>YLB</th>
<th>YUB</th>
</tr>
</thead>
<tbody>
<tr>
<td>NW</td>
<td>N.XLB</td>
<td>N.XLB + ( \frac{w}{2} )</td>
<td>N.YLB + ( \frac{w}{2} )</td>
<td>N.YLB + ( w )</td>
</tr>
<tr>
<td>SW</td>
<td>N.XLB</td>
<td>N.XLB + ( \frac{w}{2} )</td>
<td>N.YLB</td>
<td>N.YLB + ( \frac{w}{2} )</td>
</tr>
<tr>
<td>NE</td>
<td>N.XLB + ( \frac{w}{2} )</td>
<td>N.XLB + ( w )</td>
<td>N.YLB + ( \frac{w}{2} )</td>
<td>N.YLB + ( w )</td>
</tr>
<tr>
<td>SE</td>
<td>N.XLB + ( \frac{w}{2} )</td>
<td>N.XLB + ( w )</td>
<td>N.YLB</td>
<td>N.YLB + ( \frac{w}{2} )</td>
</tr>
</tbody>
</table>

Here, \( w \) denotes the width of the region represented by \( N \).
Insertion in MX-Quadtrees

(a) (b) (c) (d)
Insertion in MX-Quadtrees

After insertion of A

After insertion of B

After insertion of C

After insertion of D
Deletion in MX-Quadtrees

- Deletion in an MX-quadtree is a fairly simple operation, because all points are represented at the leaf level.
- If $N$ is an interior (i.e. non-leaf) node in an MX-quadtree whose root is pointed to by $T$, then the region implicitly represented by node $N$ contains at least one point that is explicitly contained in the tree.
- If we wish to delete a point $(x, y)$ from tree $T$, we try to preserve this property.
- This can be done as follows.
  
  * First, we set the appropriate link of $N$’s parent to NIL.
  * We then check if all the four link fields of $M$ are NIL.
  * If so, we examine $M$’s parent (let us call it $P$ for now). As $M$ is $P$’s child, we find a link field $\text{dir1}$ such that $P.\text{dir1} = M$. We then set $P.\text{dir1} = \text{NIL}$ and then (as before) check to see if $P$’s four link fields are all NIL.
  * So, we continue this process.
- Total time required for deletion is $O(k)$. 
Range Queries in MX-Quadtrees

Handled in exactly the same way as for point quadtrees. But there are two differences:

- The content of the XLB, XUB, YLB, YUB fields is different from that in the case of point quadtrees.
- As points are stored at the leaf level, checking to see if a point is in the circle defined by the range query needs to be performed only at the leaf level.
R-Trees

- Used to store \textit{rectangular regions} of an image or a map such as those shown below.
- R-trees are particularly useful in storing very large amounts of data on disk.
- They provide a convenient way of minimizing the number of disk accesses.
R-Trees

- Each $R$-tree has an associated *order*, which is an integer $K$.
- Each nonleaf $R$-tree node contains a set of at most $K$ rectangles and at least $\lfloor K/2 \rfloor$ rectangles (with the possible exception of the root).
- Intuitively, this says that each nonleaf node in the $R$-tree, with the exception of the root, must be at least “half” full.
- This feature makes $R$-trees appropriate for disk based retrieval because each disk access brings back a page containing several (i.e. at least $\frac{K}{2}$ rectangles).

R-trees manipulate two kinds of rectangles:

- “Real” rectangles (such as those shown in the map on the previous slide) or
- “Group” rectangles such as those shown below.
Ch. 4  Multidimensional Data Structures

These slides may not be duplicated without explicit written permission from Morgan Kaufmann Press.

Principles of Multimedia Database Systems  Morgan Kaufmann  Copyright ©1997
Example R-Tree

This is an R-tree of order 4, associated with the rectangles shown earlier.

R-tree nodes have the following structure:

\[
\text{rtnodetype} = \text{record} \\
\quad \text{Rec}_1, \ldots, \text{Rec}_K : \text{rectangle} ; \\
\quad P_1, \ldots, P_K : \uparrow \text{rtnodetype} \\
\text{end}
\]
Insertion into an R-Tree
Insertion into an R-Tree
An Incorrect Insertion into an R-Tree
Deletion in R-Trees

- Deletion of objects from R-trees may cause a node in the R-tree to “underflow” because an R-tree of order $K$ must contain at least $\lceil K/2 \rceil$ rectangles (real or group) in it.
- When we delete a rectangle from an R-tree, we must ensure that that node is not “underfull.”

**Example:** Delete R9 from the following R-tree.

![Diagram of R-tree](image)
Deletion in R-Trees

- If we delete R9, then the node containing rectangle R9 would have only one node in it.
- In this case, we must create a new logical grouping.
- One possibility is to reallocate the groups as follows:

<table>
<thead>
<tr>
<th>Group</th>
<th>Rectangles</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>R1, R2, R3</td>
</tr>
<tr>
<td>G2</td>
<td>R4, R6, R7</td>
</tr>
<tr>
<td>G3</td>
<td>R5, R8</td>
</tr>
</tbody>
</table>

- The new new R-tree is:

```
  G1 G2 G3
 / | | |
 / | | |
 / R1 R2 R3 / R4 R6 R7 / R5 R9
  | | |   | | |
  | | |   | | |
  | | |   | | |
  | | |   | | |
  | | |   | | |
```

Leaf Nodes