COMP30110 - Spatial Information Systems

Spring Trimester Assessment

Michael Mullin, 17360573

Question 1:

Raster and vector are two very different but common data formats used to store geospatial data. Raster data is represented by partitioning the data into polygonal units of equal size called cells. An example of raster data is a satellite image of a city. Vector data consists of primitive spatial data objects which are located by Cartesian co-ordinates in a spatial reference frame. An example of vector data is a geographic map of a city.

Data Models

The vector data model assumes that the earth's surface is composed of discrete objects such as rivers, lakes, country borders, and land parcels. The model is broken down into three primitive spatial data objects: points, lines, and polygons. A point uses a single co-ordinate pair to define its location, while a line is defined by an ordered set of co-ordinates, and a polygon is formed by a set of connected lines where the start and endpoint have the same co-ordinate.

The raster data model forms a grid-like structure that holds values at regularly spaced intervals over the extent of the raster. These intervals that make up the grid-like structure are called cells. Each cell has two associated values: a positional value that marks its identity, and attribute values of the underlying area it represents.

Storage

Vector data consists of spatial entities and the relations between those spatial entities. The relations stored are vertex-based, edge-based, and face-based. The spatial entities stored are edges, vertices, and faces. Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterators. Vector data structures can be evaluated in terms of space complexity (memory needed) and time complexity of the algorithms for calculating connectivity relations.

Raster datasets record the values for each cell within an image and can sometimes be very large in size. The simple raster data storage techniques are inefficient as they use up the same amount of disk space, regardless of the data distribution. Compression methods such as Quadtrees can be used to efficiently store the data.

Manipulation Strategies

Vector data can be queried, analysed, and updated using an extension of SQL. An algorithm must iterate through the stored spatial entities and their relations to manipulate the data. Vector data can be converted to the raster data type by overlaying the vector data on a raster array. This conversion often results in a stair-step distortion (aliasing).

Raster data can be scanned using multiple different methods such as the row method, row-prime method, Morton scan method, and the Peano-Hilbert curve method. Raster data can be converted to the vector data type by simply connecting adjoining pixels orthogonally and diagonally (8-connected).

Applications

The three primitive vector data objects (points, lines, and polygons) all work together to produce the vector data type. Large scaled maps are formed by the vector data type where points can represent city names, where lines can represent roads/rivers, and where polygons can represent areas such as a boundary of a county.

Raster data is exceptionally useful for a wide range of applications including orthoimages (from an aircraft), satellite imagery, and scanned maps. Raster data can also represent thematic maps by assigning different cells to different colors based on the theme being used.

Question 2:

A planar graph is one that can be drawn in the Euclidean plane in such a way that its edges do not intersect each other, except at their endpoints. The embedding of such a planar graph in the Euclidean plane is called a plane graph. A straight-line plane graph defines a partition of the plane into a collection of simply connected polygonal regions called faces. This partition is called a plane subdivision.
Plane subdivisions contain three kinds of topological entities: vertices, edges, and faces. A vertex represents a single point on the plane. An edge connects two, not necessarily distinct, vertices. A face represents a connected subset of the plane. We can define nine ordered relations with these three entities in mind: Vertex-based (vertex-vertex, vertex-edge, vertex-face), edge-based (edge-edge, edge-vertex, edge-face) and face-based (face-face, face-vertex, face-edge).
Topology is a set of rules that represent the relationships between neighbouring points, lines, and polygons. The inclusion of topological information allows for efficient defining and analysing of the spatial relationships within a dataset. Topological data structures incorporate some connectivity between spatial objects by explicitly storing a subset of the nine relations mentioned above. The only topological relation possible is 'meet' which means two entities share a portion of their boundary.
The doubly-connected edge list (DCEL) topological data structure was developed by computational geometry researchers Preparata and Shamos in 1985. The DCEL stores the three topological entities (V, E, and F) but does not store the full nine ordered relations. It stores all edge-based relations (EV, EE, and EF) and two other partial relations (FE*, VE*). The partial and full relations can be used to create the relations that are not explicitly stored within the data structure, e.g. Relation VE can be obtained by combining VE* and EE. Several other data structures have been proposed that differ in the entities and relations they store such as the symmetric data structure (Woo, 1985), the arc-node data structure (Peucker and Chrisman, 1975), and the Winged-edge data structure (Weiler, 1985). Some plane subdivisions contain faces that are all triangles. In this case, the entities stored are vertices, edges, and triangles. We can define nine ordered relations with these three entities in mind: Vertex-based (VV, VE, VT), Edge-based (EE, EV, ET), and Triangle-based (TT, TV, TE). A summary of common topological data structures can be seen below:

	Entities stored	**Full relations**	**Partial relations**
DCEL	V, E, F	EV, EE, EF	FE*, VE*
Symmetric	V, E, F	EV, VE, FE, EF	
Arc-node	V, E, F	EV, EF	
Simplified symmetric	V, E, T	TE, ET, EV	VE*

One of the main advantages of using topological data structures is that they support topological queries. The simplest non-topological data structure (spaghetti data structure) does not support topological queries and therefore is quite inefficient. Topological data structures also provide important criteria which can be used to assess the integrity and consistency of geographic databases. The data set can be queried to detect some common errors such as country boundaries that overlap or country boundaries that aren't fully closed. Despite the efficiency that topological data structures bring, they are static in nature and it can be time-consuming to define topology for large datasets. This static nature implies that every time the dataset is edited, the topology of the dataset must be rebuilt, e.g. if a vertex co-ordinate is changed.

Question 3:

The amount of spatial data has increased rapidly in recent years giving rise to the essential need for powerful systems to store and manipulate this data. These powerful systems must be capable of correctly storing both the spatial (e.g x,y co-ordintates) and non-spatial (e.g location name) aspects of the data. The original Geographic Information Systems (GIS) completely relied on file-based systems which were not feasible. The Database Management Systems (DBMS) did not directly support spatial queries and required a join of several different tables which was highly inefficient. The DBMS inefficiency and the exponential growth of spatial data gave rise to the new *Loosely coupled approach* architecture of more recent GIS systems.

This approach relies on the partition of spatial and non-spatial data which are handled by two separate but linked co-existing systems. The architecture contains a DBMS system where the non-spatial aspects are stored (using SQL for queries) and a spatial data file system where the spatial aspects of the data are stored in vector files. The objects in the vector files are linked to tables in the database corresponding to attribute information. Having a dedicated model for dealing with geospatial data is a significant advantage of this approach as it allows for proper geospatial data management. This approach also supports spatial queries. However, having two co-existing systems which are quite different in nature is complex and makes it difficult to integrate heterogeneous models within the same system. This complexity requires a lot of training to be able to exploit the sophisticated functionality.

Within the last 20 years, a Spatial Database Management System (SDBMS) has been developed which relies on the *Integrated approach*.

This approach stores all aspects of the data (non-spatial and spatial) in the same database. This is very different from the loosely coupled approach where there is complete separation between the spatial and the non-spatial aspects of the data stored in two separate systems. This new approach significantly reduces the complexity of system management by eliminating the hybrid-based architectures that are present in the loosely coupled approach. The integrated approach also allows for better data management and data integration as all aspects (both spatial and non-spatial) are stored in a single database. The integrated approach has been adopted by many database vendors such as Oracle Spatial and Graph, Postgres/PostGIS.

Oracle Spatial adapts the integrated approach by using the object-relational model for representing geometries. This model stores the geometries (spatial aspects of the spatial objects) in a single column of object type SDO_GEOMETRY. The SDO_GEOMETRY object type defines the following attributes:

1. SDO_GTYPE
   - Defines the type of geometry (point, line, polygon)
2. SDO_SRID
   - Defines the co-ordinate system
3. SDO_POINT
   - Defines the co-ordinates for a point geometry
4. SDO_ELEM_INFO
   - Stores the co-ordinate values for the boundary of a geometry as an array
5. SDO_ORDINATES
   - Gives information on how to interpret the ordinates

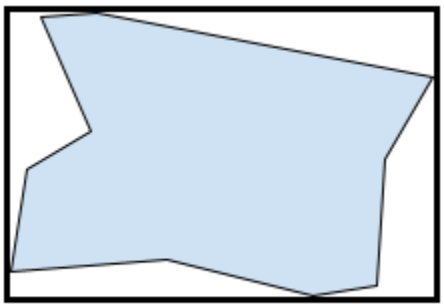
This object type enables spatial data to be stored, accessed, and analysed quickly and efficiently in the database.

ORACLE®

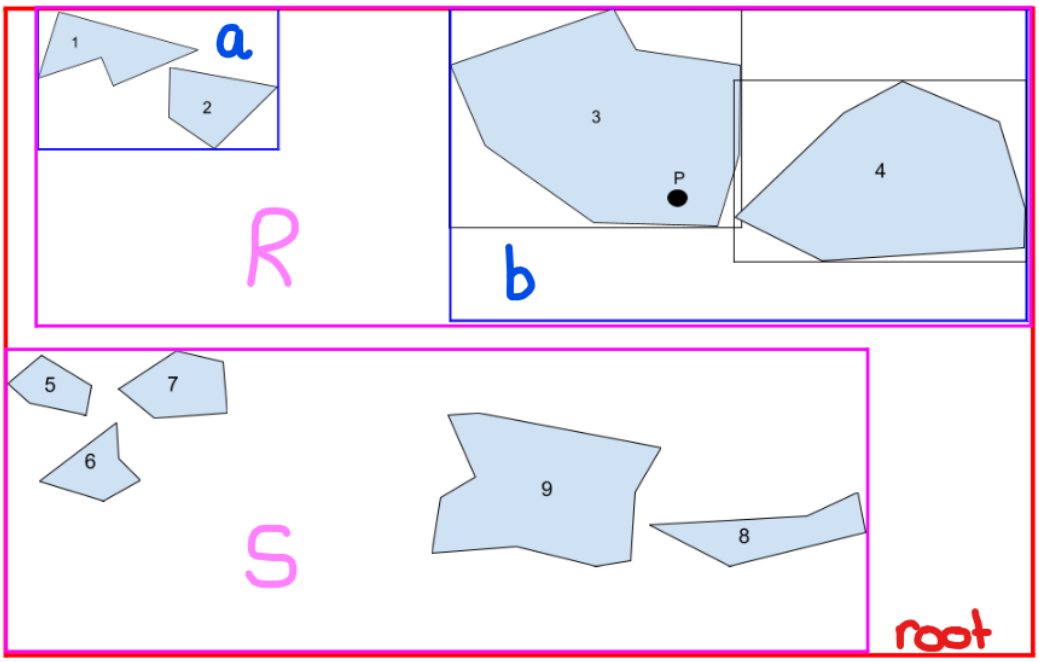
SPATIAL

Question 6:

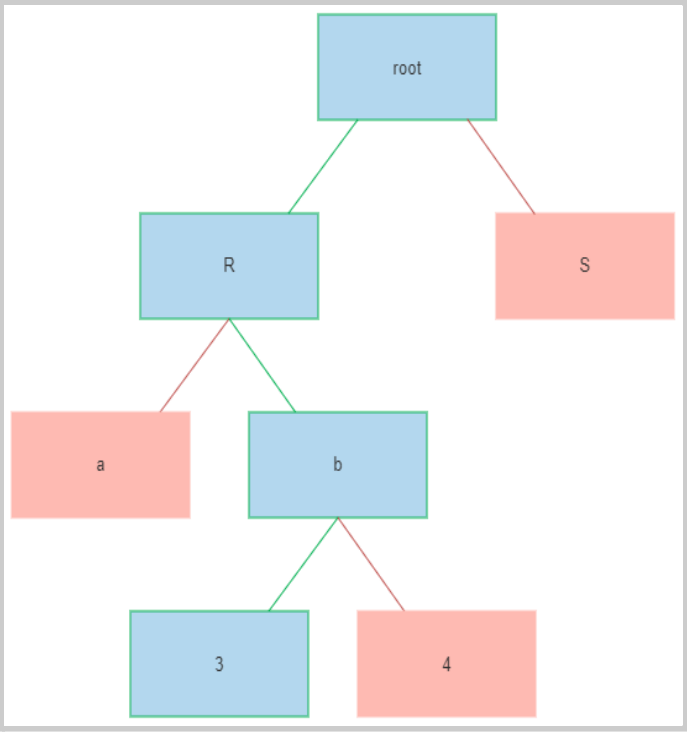
Spatial Indexing is a mechanism that allows you to organise data in a way that supports and speeds up specific spatial queries. Spatial indexing can be performed on vector data which is a type of spatial data that consists of primitive spatial data objects which are located by Cartesian co-ordinates in a spatial reference frame. The three primitive spatial data objects are points, lines, and polygons. Without spatial indexing, any database search would require sequentially scanning all objects which involves a high number of disk accesses and can be computationally expensive. Spatial Indexing speeds up searching by organizing the data into a search tree which can be quickly traversed to find a particular record.
In the case of collections of polygons, an approximation of the geometry is created by using the minimum bounding rectangle (MBR). The MBR of a geometry is the bounding geometry formed by the minimum and maximum (X, Y) co-ordinates (see fig below). By using the MBR for building the spatial index, the cost of evaluating expensive geometric predicates is greatly reduced.



There are two types of indexes implemented in Oracle Spatial: R-trees and Quadtrees. R-trees are tree data structures used for indexing multi-dimensional information such as geographical co-ordinates, rectangles, or polygons. The key idea behind R-trees is to group nearby objects and represent them with their MBR. Multiple smaller MBRs are grouped into larger MBRs to form intermediate nodes in the tree and this process is repeated until one rectangle is left that contains everything. An example of executing a point query (to find which polygon contains point P) using an R-tree can be seen below.
Dataset:



The R-Tree is traversed from top to bottom in the following way for the point query point P:



- Point P is contained by root.

- The children of root are checked, R and S. Object S does not contain point P, therefore the entire subtree of S will be disregarded. The subtree of object R is continued.

- The children of R are checked, a and b. Object a does not contain the point P, therefore the entire subtree of a is disregarded. The subtree of object b is continued.

- The children of b are checked, 3 and 4. Object 4 does not contain the point P, we find the MBR of object 3 contains the point P.

- The actual geometry of object 3 is then checked for the inclusion of point P.

One of the main advantages of executing a point query using the approach outlined above is that it uses MBRs which are not computationally expensive in comparison to using the actual geometry of each object. The actual geometry of an object is only used when the R-tree traversal reaches the leaf node (object 3 in the example above) to check for the inclusion of the point in question.
There have been some variations of R-trees created including R+ trees and R* trees. R+ trees avoid overlapping rectangles in the intermediate nodes of the tree. R* trees were proposed to improve the tree structure at the cost of longer insertion operations.