



Cairo University



# **Geographic Information Systems And Spatial Databases**

Course Code: IS443

**Dr. Mohamed Nour Eldien**  
**Information systems Department**  
**Faculty of Computers & Artificial Intelligence**  
**Cairo university**

# Course Description

## 1. Course Overview

Geographic Information Systems (GIS) are tools for managing, describing, analyzing, and presenting information about the relationships between where features are (location, size and shape) and what they are like (descriptive information known as attribute data). Because its techniques allow one to represent social and environmental data as a map, GIS has become an important tool across a variety of fields including planning, architecture, engineering, public health, environmental science, epidemiology, and business.

Further, GIS has become an important political instrument allowing communities and regions to (geo)graphically tell their stories. GIS is a powerful tool, and this course is meant to introduce students to the basics. Because GIS can be applied to many research fields, this class is meant to give students an understanding of its possibilities along with the capabilities to begin engaging those possibilities. The class will focus on teaching through practical example.. Roughly speaking, the course is organized in two parts. The first half of the course will focus on the basics by leading the students through skills-based GIS exercises. The second half of the course will be focused on individual student projects for which each student will be required to find data and design the methods of analysis to be used based on the techniques learned in the course.

# Course Description

## 2. Objectives

The course seeks to provide students with a basic level of familiarity with several aspects of Geographic Information Systems and Geographic Information Science, such that the range of possibilities for GIS-based work is understood and an adequate foundation for engaging those possibilities is laid. Thus, the objectives for the course are:

- Providing an understanding of basic skills necessary to work with GIS, predominantly using ESRI's ArcGIS software
- Introducing students to software and techniques beyond ESRI products
- Teaching spatial data visualization techniques along with introductory knowledge of effective cartography and additional software for the production of maps and other information graphics
- Teaching skills needed to develop and execute a project requiring GIS as a management, analytical, and/or visualization tool
- Identifying and accessing publicly available data sets
- Teaching the skills necessary to create GIS data through a variety of methods including those offered by global positioning system (GPS) technologies
- Providing an introductory understanding of the ethical questions surrounding data creation, analysis, and representation

# Course Description

## 3. Table of content

### Chapter 1 : Introduction to GIS

- 1.1. What is GIS?
- 1.2. GIS Concepts
- 1.3. GIS Components
- 1.4. GIS Data Types
- 1.5. GIS data models
- 1.6. Maps and Cartography
- 1.7. Basic Elements of Map Composition
- 1.8. Projections and Coordinate Systems

### Chapter 2 :Spatial Databases

- 2.1. Vector Database
- 2.2. Raster Database

### Chapter 3:Spatial Query Languages

- 3.1. Standard Database Query Languages
- 3.2. Relational Algebra
- 3.3. Basic SQL Primer
- 3.4. Extending SQL for Spatial Data
- 3.5. Example Queries that emphasize spatial aspects
- 3.6.Trends: Object-Relational SQL

### Chapter 4: Spatial Networks

- 4.1. Example Network Databases
- 4.2. Conceptual, Logical and Physical Data Models
- 4.3. Query Language for Graphs
- 4.4. Graph Algorithms



Cairo University



# Geographic Information System And Spatial Databases

Course Code: IS443

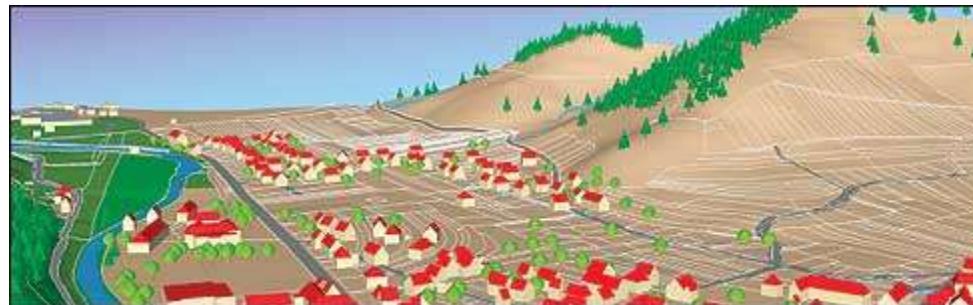
## Chapter 1: Introduction to GIS

# What is GIS?

**GIS = Geographic Information System(s)**

*GIS is a collection of computer hardware, software, and geographic data for capturing, managing, analyzing, and displaying all forms of geographically referenced information.*

**Environmental Systems Research Institute (ESRI), 2007**

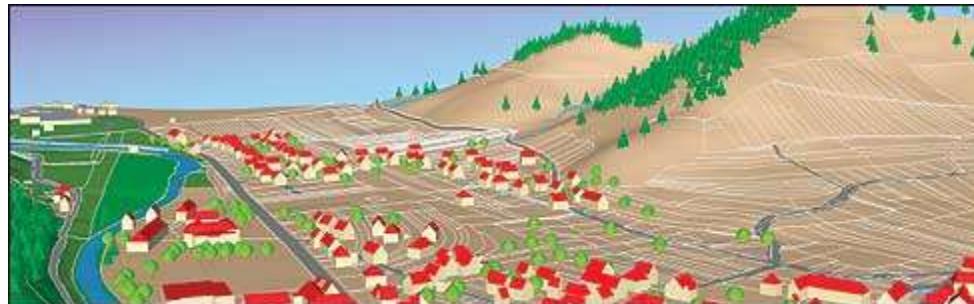


# 1.1. What is GIS?

**GIS = Geographic Information System(s)**

*GIS is a collection of computer hardware, software, and geographic data for capturing, managing, **analyzing**, and **displaying** all forms of geographically referenced information.*

**Environmental Systems Research Institute (ESRI), 2007**

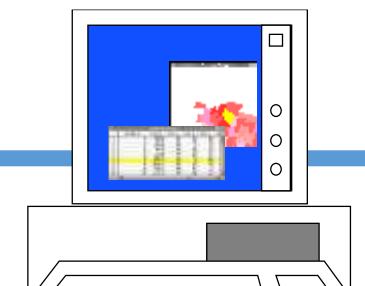


# Defining Geographic Information Systems GIS

**'A GIS is designed for the collection, storage, and analysis of objects and phenomena where geographic location is an important characteristic or critical to the analysis.'** **Stanley Aronoff**

**"Computer tool for managing geographic feature location data and data related to those features."** **Allan B. Cox**

**GIS is a tool for managing data about **where** features are (geographic coordinate data) and **what** they are like (attribute data), and for providing the ability to **query**, **manipulate**, and **analyze** those data.**



**G | S**

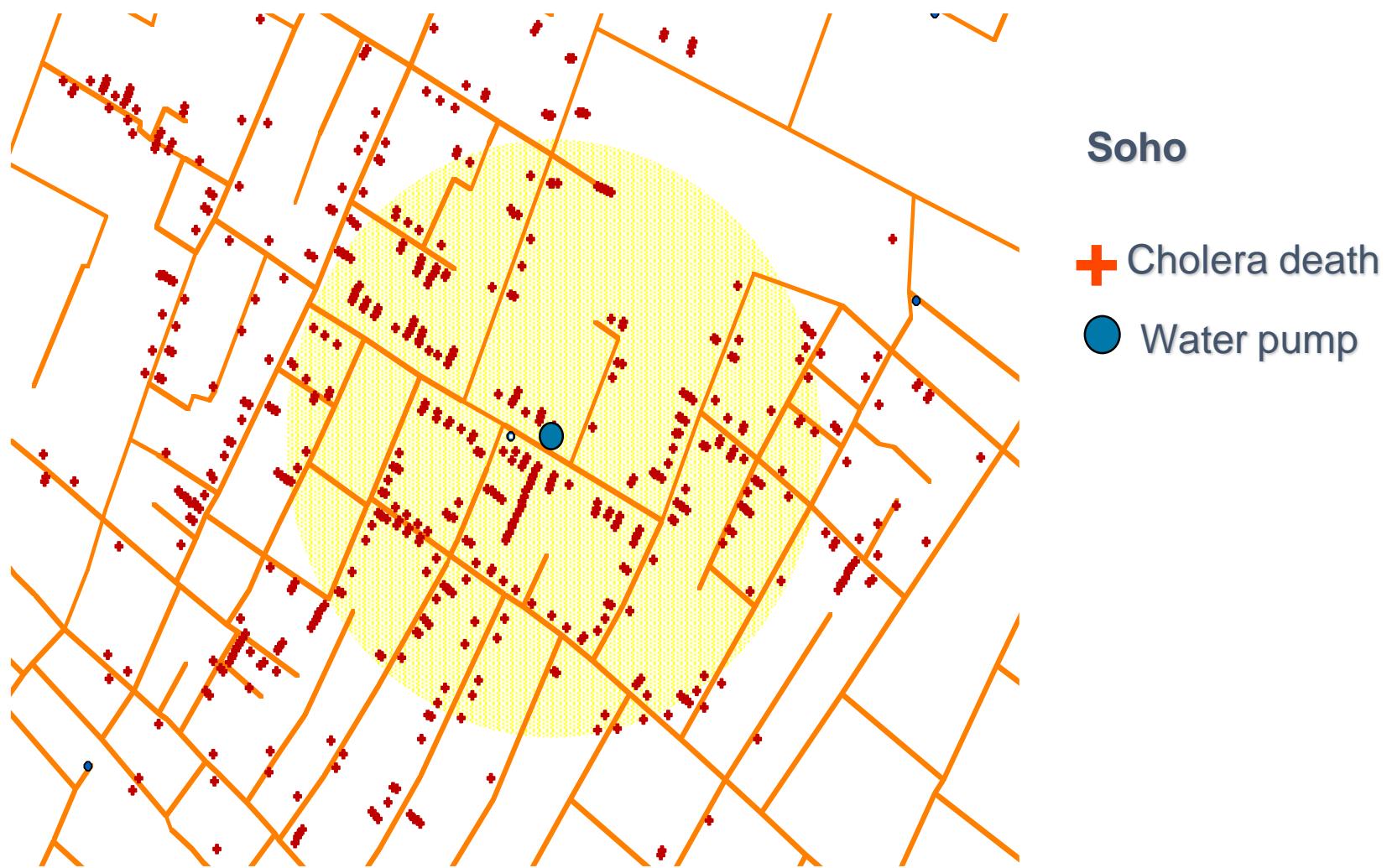
ELOAROAD Browser		
DESC	FCODE	ID
Pall Mall East	20.61	
Pall Mall	20.61	
St James's Street	20.61	
Piccadilly	20.61	
Waterloo Place	20.61	
Charles II Street	20.61	
Shaftesbury Avenue	20.61	
Great Windmill Street	20.61	

# GIS concepts are not new!

- London cholera epidemic 1854

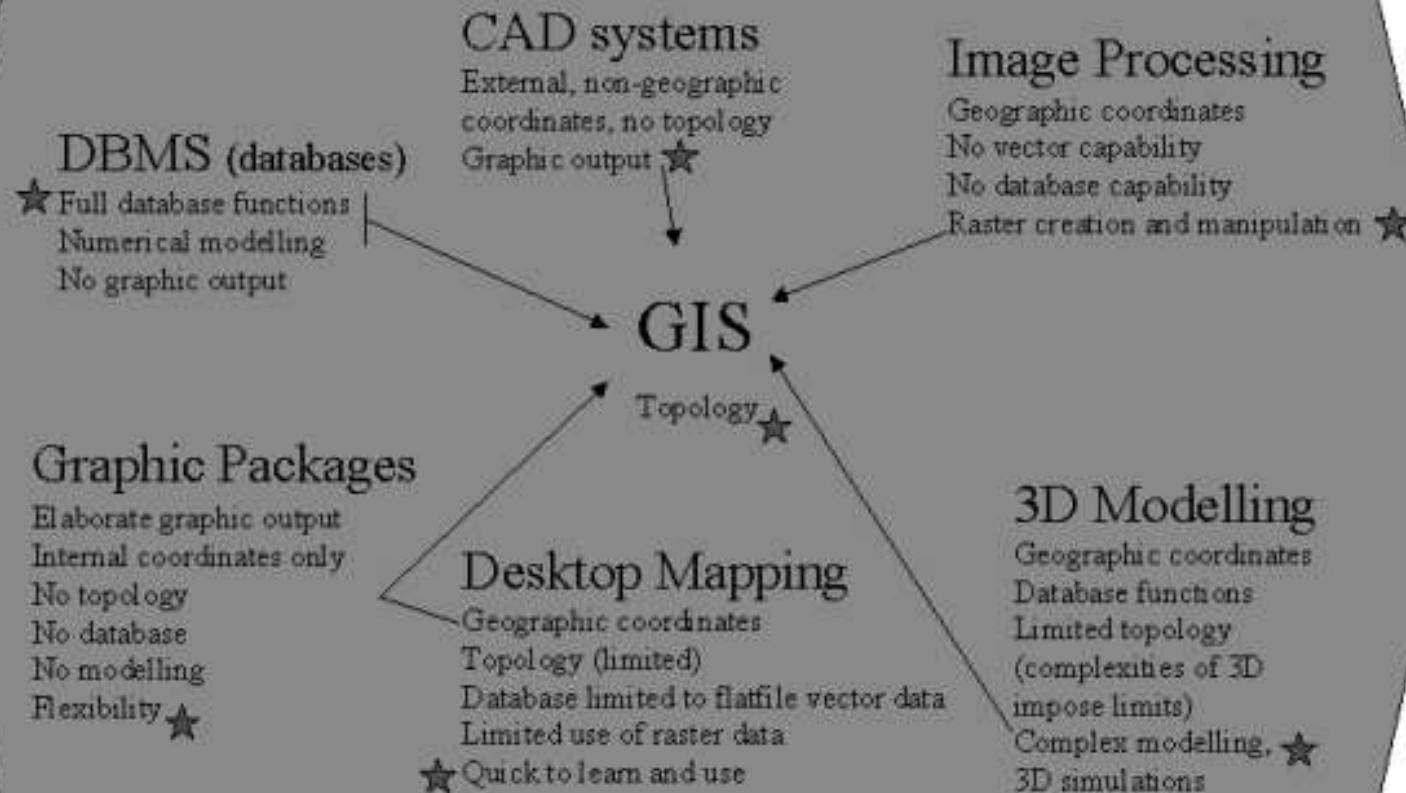


# Spatial information handling 1854



# GIS and Related Software

## GIS in a group of related software



★ Primary strength

## 1.2. GIS Concepts

This section covers the two basic GIS concepts you need to know to effectively use any GIS maps

### GIS Concept #1: Features have attributes associated with them.

Imagine a tree. How would you keep track of and communicate information about this tree to other people who need to know all about it? You might use a database to keep track of what species it is, how old it is, how tall it is, how healthy it is, and any other attributes that are important. This tree is one **record** in a **database**. We call each category (i.e. tree height) a **field**.



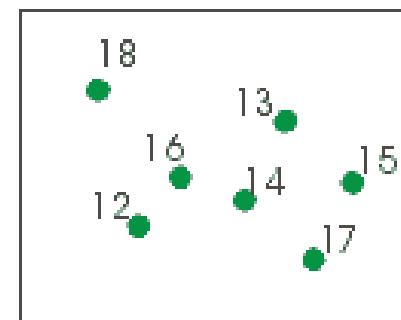
ID	12
Type	Cedar
Age	110
Height	67

Now imagine a grove of trees that you need to keep track of attributes for. Because we are now dealing with more than one tree, it becomes relevant *where* each tree is so we know what information relates to which tree



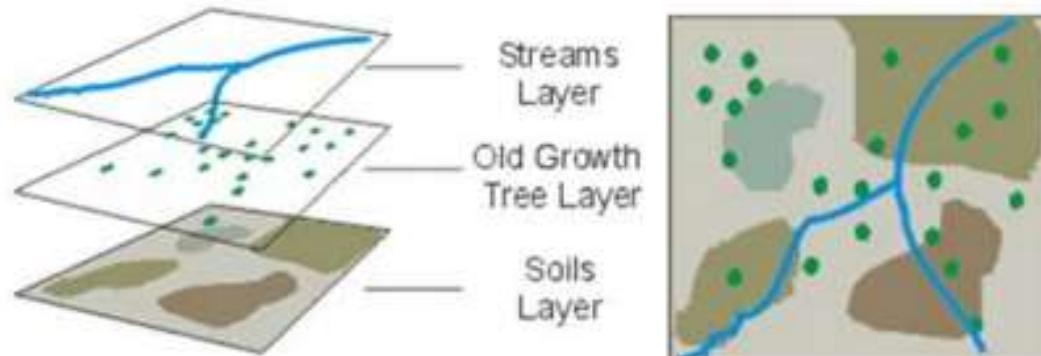
ID	Type	Age	Height
12	Cedar	110	67
13	Pine	135	80
14	Cedar	120	72
15	Cedar	120	70
16	Spruce	80	65
17	Spruce	75	60
18	Pine	125	73

We map the location of each tree and identify which attributes belong to which tree. This is the foundation of GIS. A GIS tells us where something is and what it is. Computers are synonymous with GIS, and using a computer we can have hundreds of fields (different attributes) for millions of records (trees).

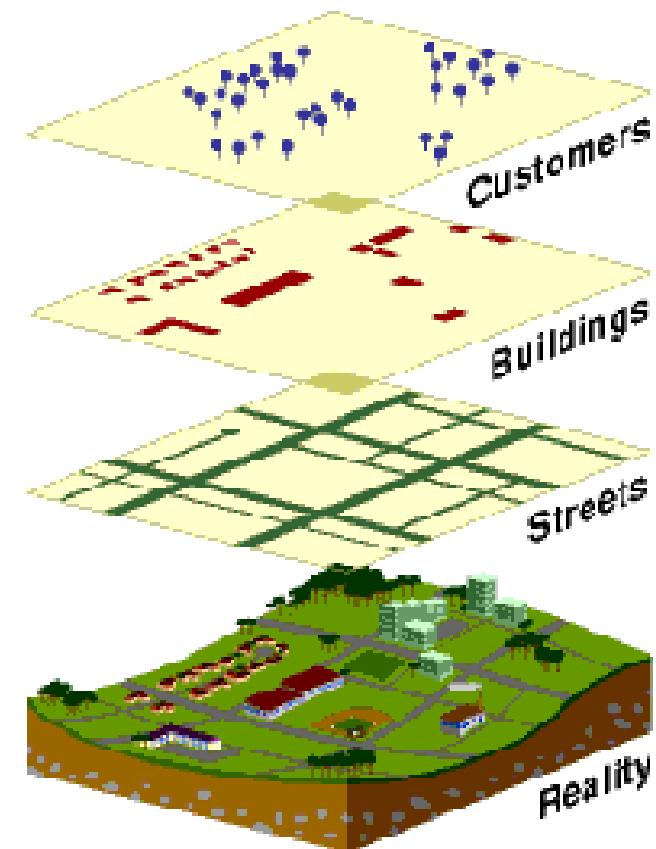


## GIS Concept # 2: Information is separated into layers.

We can also have other layers of information in our GIS. Our information on trees would constitute one layer of information. Each tree is a unique feature in that layer. We could also have a layer with rivers and a layer with soil types. The rivers layer contains river features. The soil type layer contains soil classification features.



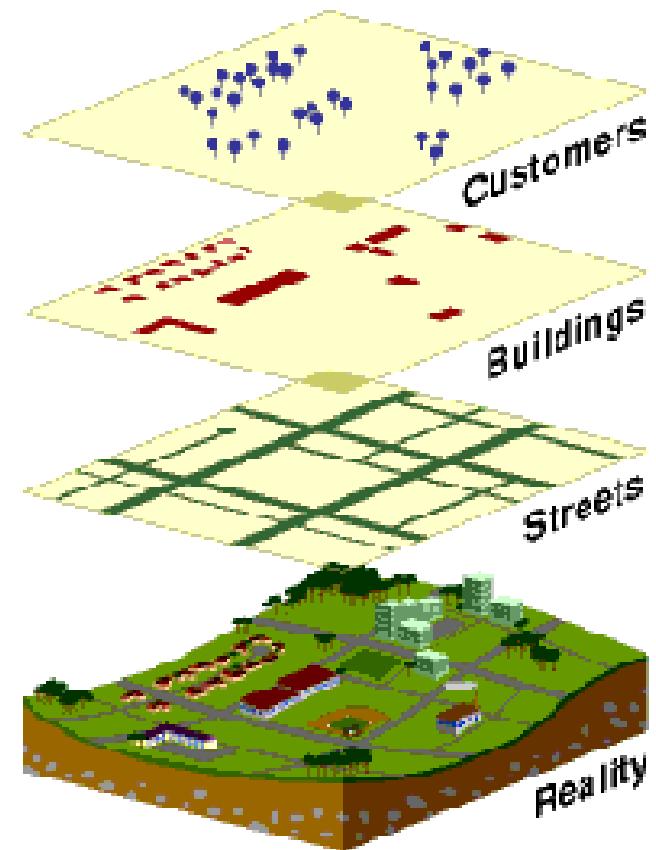
You might hear people talk about **coverage**, **Geodatabase**, or **shapefile**. All these terms are other names for **layers** of information.



*Layers representing the real world*

## How GIS Works

A GIS stores information about the world as a collection of thematic layers that can be linked together by geography. This simple but extremely powerful and versatile concept has proven invaluable for solving many real-world problems from tracking delivery vehicles, to recording details of planning applications, to modeling global atmospheric circulation.



*Layers representing the real world*

# 1.3. GIS Components

- ❑ A working GIS integrates five key components: hardware, software, data, people, and methods.

- Hardware
- Software
- Data
- People
- Methods

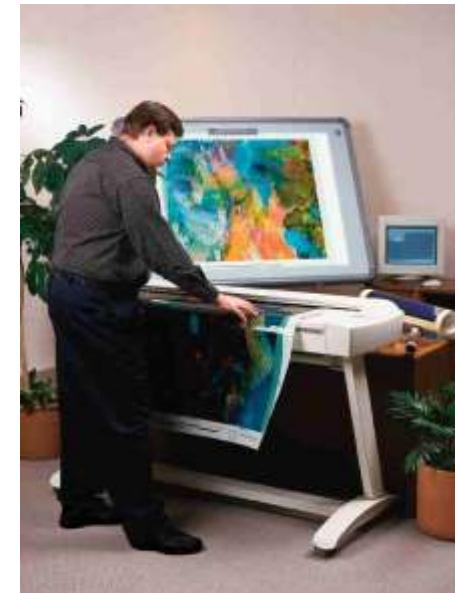


## 1.3.1. Hardware

❑ Hardware is the computer on which a GIS operates, including the resources available to the computer:

- printers
- plotters
- digitizers
- scanners
- monitors
- network
- wide area communications

❑ Today, GIS software runs on a wide range of hardware types, from centralized computer servers to desktop computers used in stand-alone or networked configurations.



## 1.3.2. Software

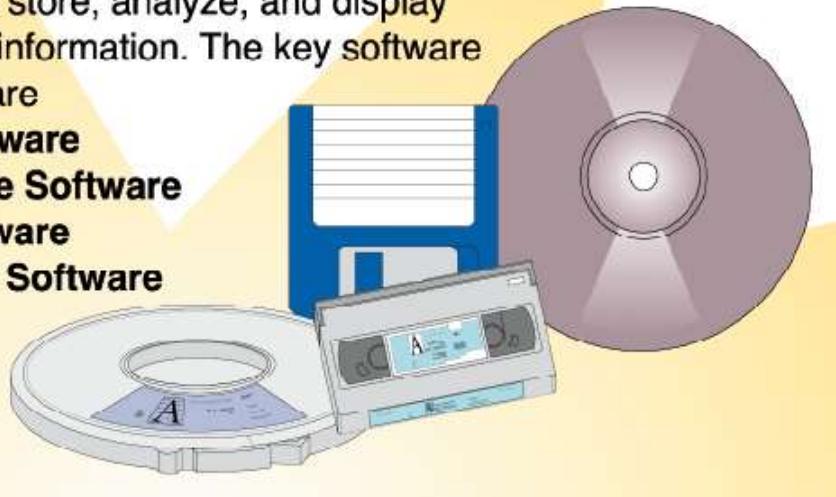
- GIS software provides the functions and tools needed to
  - store
  - query
  - display
  - analyze
  - create
  - Modify data.



# SOFTWARE

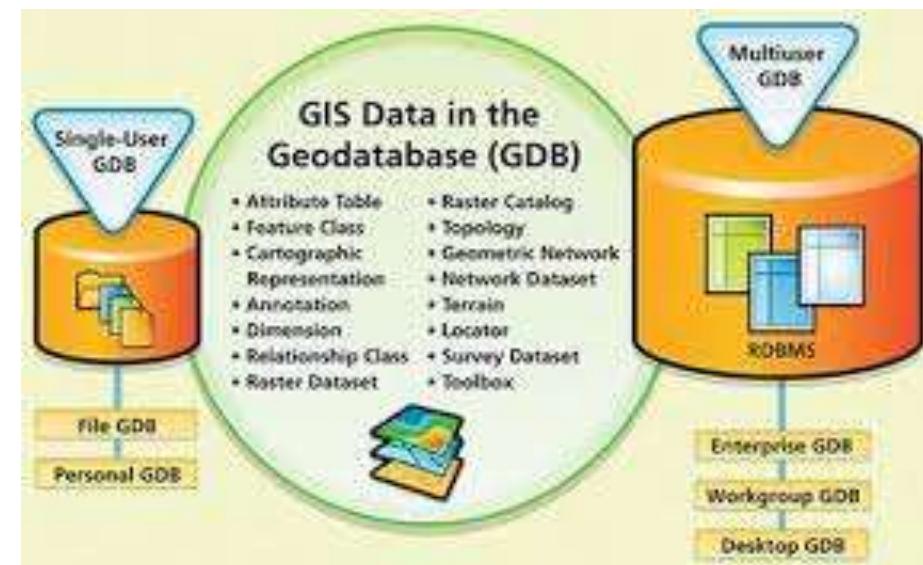
GIS software provides the functions and tools users need to store, analyze, and display geographical information. The key software components are

- **GIS Software**
- **Database Software**
- **OS Software**
- **Network Software**



### 1.3.3. Data

- Possibly the most important component of a GIS is the data.
- Geographic data and related tabular data can be collected in-house or purchased from a commercial data provider.
- A GIS will integrate spatial data with other data resources and can even use a DBMS, used by most organizations to organize and maintain their data, to manage spatial data.
- First, **GIS** uses **data** that contains a location or space, therefore it is displayed in a map or picture form. Recently, aerial or satellite **data** becomes more and more important as new technologies are introduced. As a location based **data**, **GIS data** is usually large-sized as is big **data**.



## 1.3.4. People



❑ GIS technology is of limited value without the people who manage the system and develop plans for applying it to real world problems. GIS users range from technical specialists who design and maintain the system to those who use it to help them perform their everyday work.

# PEOPLE

GIS technology is clearly of limited value without people to manage the system and to develop plans for applying it. Users of GIS range from highly qualified technical specialists to planners, foresters, and market analysts who use GIS to help with their everyday work.

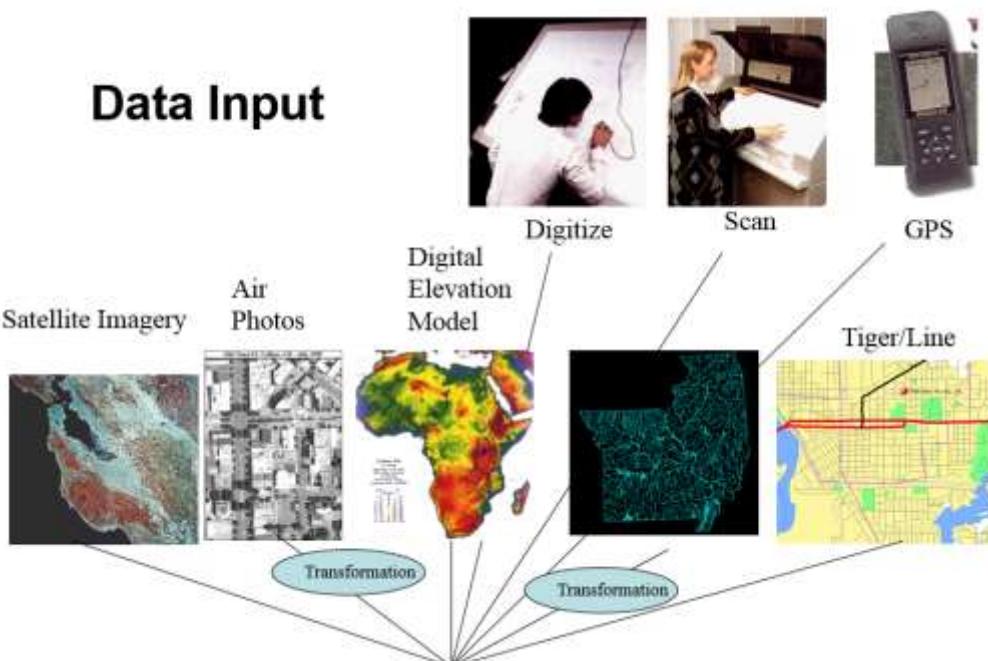
- **Administrators**
- **Managers**
- **GIS Technicians**
- **Application Experts**
- **End Users**
- **Consumers**



# 1.3.5. Methods

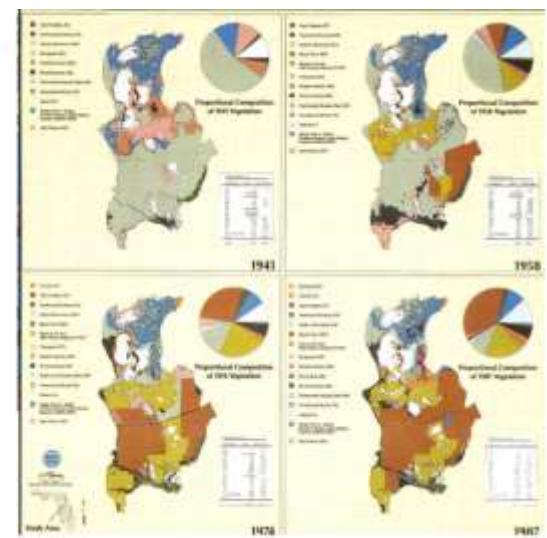
- A successful GIS operates according to a well-designed plan and business rules, which are the models and operating practices unique to each organization.
- General purpose GISs essentially perform five processes or tasks.
  - Input
  - Manipulation
  - Management
  - Query and Analysis
  - Visualization

## Data Input

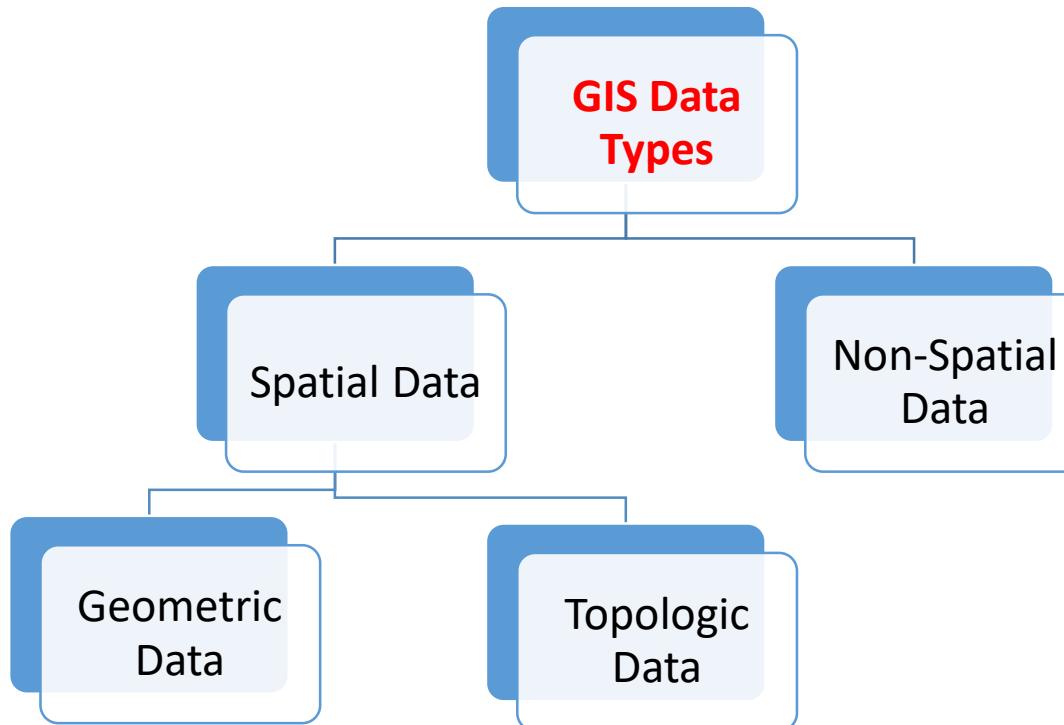


## Information Output

- Maps
- Charts
- Reports



# 1.4.GIS Data Types



Represents features that have a known **location** and **Form** on earth.

**How geographic features are related to one another and where they are in relation to one another.** Topology is the critical element that distinguishes a GIS from a graphics or automated cartography system. It is essential to the ability of a GIS to employ spatial relationships. Topology is what enables a GIS to emulate our human ability to discern and manipulate geographic relationships.

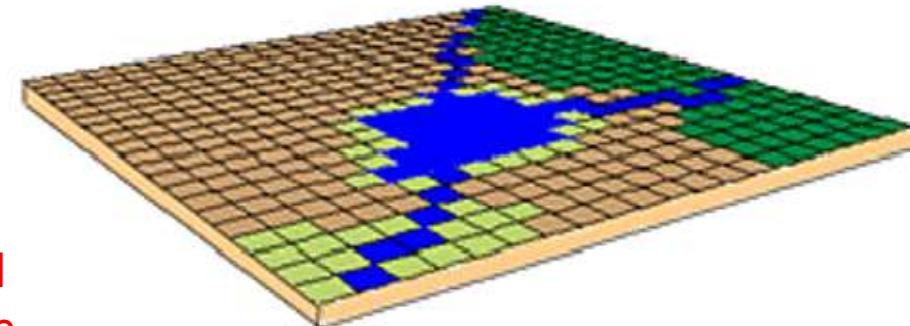
Attribute data are the information linked to the geographic features (spatial data) that describe features

# 1.5. GIS data models

- RASTER



Define space as an array of equally sized cells arranged in rows and columns. Each cell contains an attribute value and location coordinates



- VECTOR

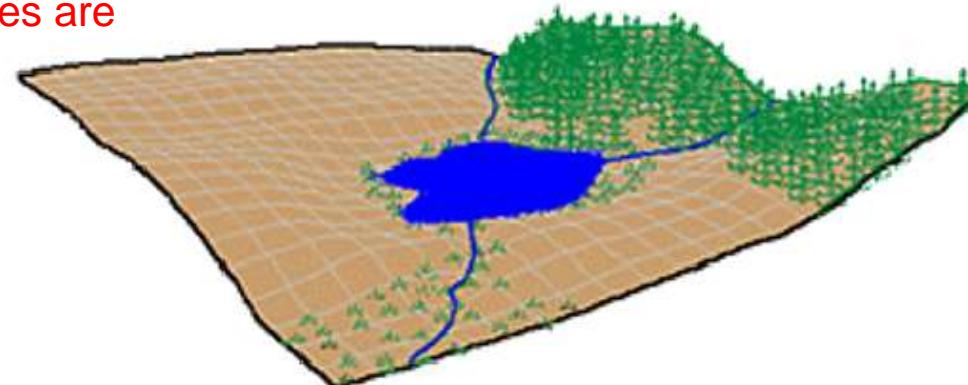


Use x-, y- coordinates to represent point, line, area, network, surface

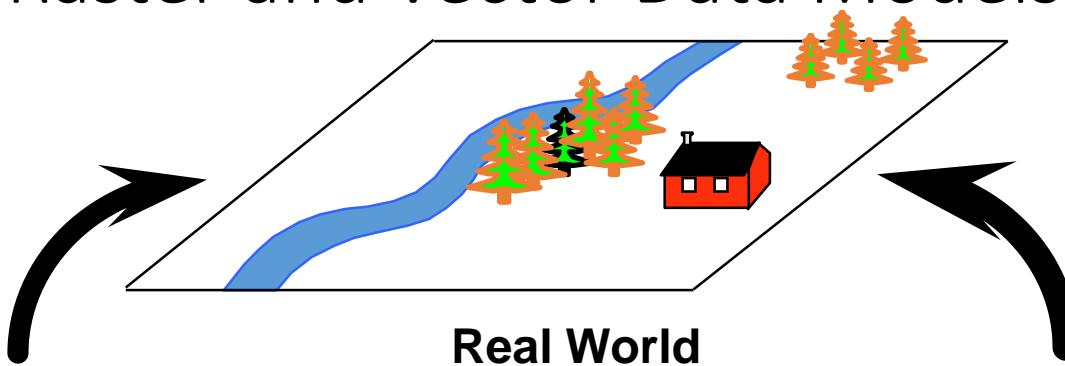
Point as a single coordinate pair, line and polygon as ordered lists of vertices, while attributes are associated with each features



- Real World

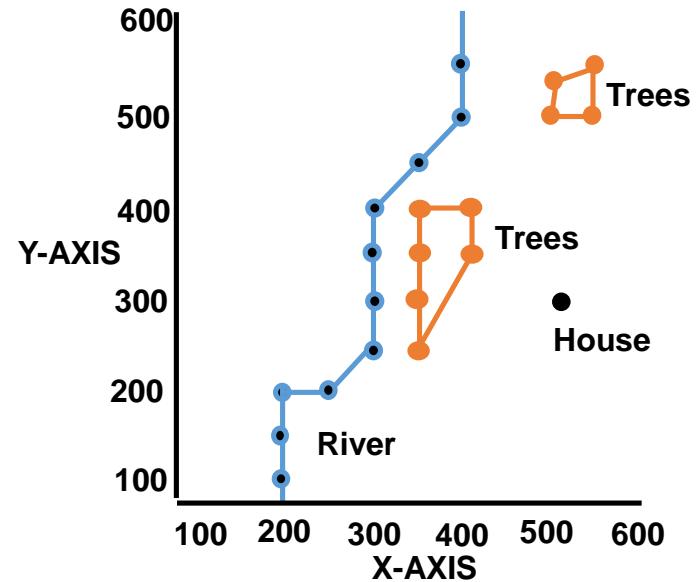


# Raster and Vector Data Models



	1	2	3	4	5	6	7	8	9	10
1						B			G	
2					B	B	G	G		
3			B							
4		B	G	G						
5		B	G	G						
6		B	G			BK				
7			B	G						
8		B	B							
9		B								
10		B								

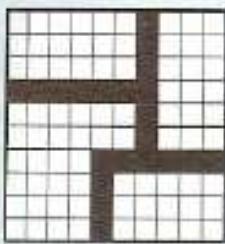
Raster Representation



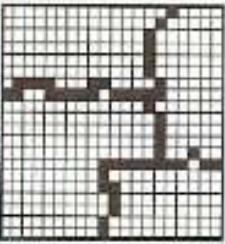
Vector Representation



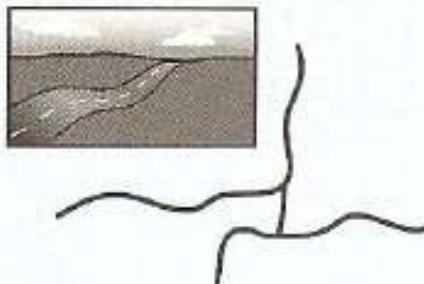
5x5 Grid



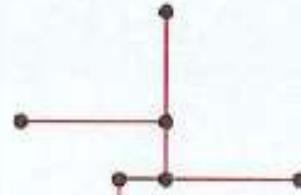
10x10 Grid



20x20 Grid



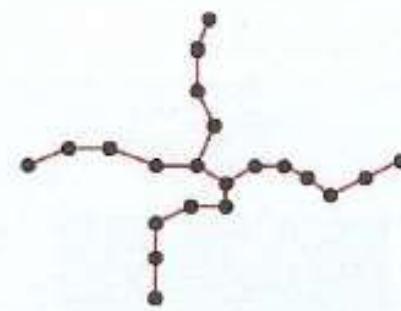
Network: roads



7 points



14 points



21 points

Effects of changing resolution and scale

# Vector – Advantages and Disadvantages

- **Advantages**
  - Good representation of reality
  - Compact data structure
  - Topology can be described in a network
  - Accurate graphics
- **Disadvantages**
  - Complex data structures
  - Simulation may be difficult
  - Some spatial analysis is difficult or impossible to perform

# Raster – Advantages and Disadvantages

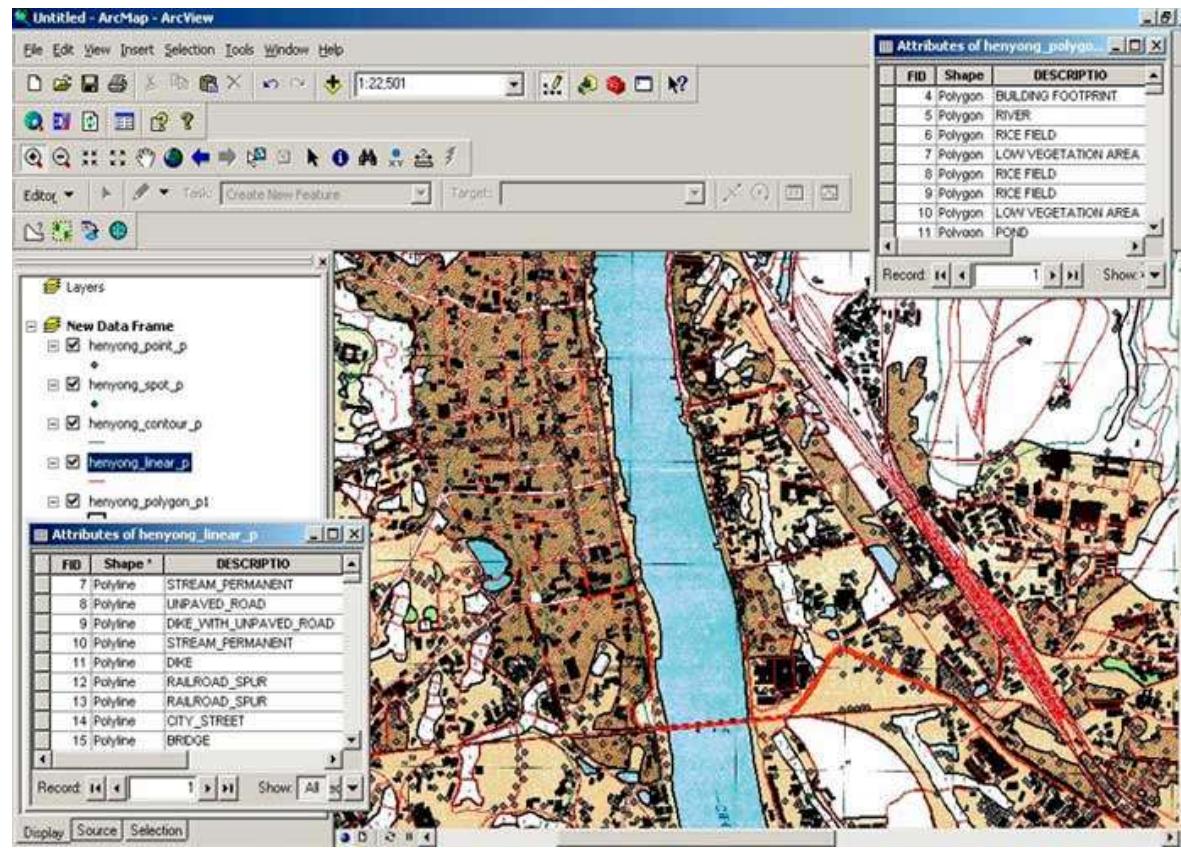
- Advantages
  - Simple data structure
  - Easy overlay
  - Various kinds of spatial analysis
  - Uniform size and shape
  - Cheaper technology
- Disadvantages
  - Large amount of data
  - Less “pretty”
  - Projection transformation is difficult
  - Different scales between layers can be a nightmare
  - May lose information due to generalization

## 1.6. Maps and Cartography

- Maps are the main source of data for GIS
- According to the International Cartographic Association, a map is:  
*“a representation, normally to scale and on a flat medium, of a selection of material or abstract features on, or in relation to, the surface of the Earth”*
- The term "map" is often used in mathematics to convey the notion of transferring information from one form to another, just as cartographers transfer information from the surface of the Earth to a sheet of paper
- The term "map" is used loosely to refer to any visual display of information, particularly if it is abstract, generalized or schematic

# Types of maps

1. **Topographic maps** - are a detailed record of a land area, giving geographic positions and elevations for both natural and man-made features. They show the shape of the land the mountains, valleys, etc.
2. **Thematic map** - a tool to communicate geographical concepts such as the distribution of population densities, climate, movement of goods, land use etc.



## 1.7. Basic Elements of Map Composition

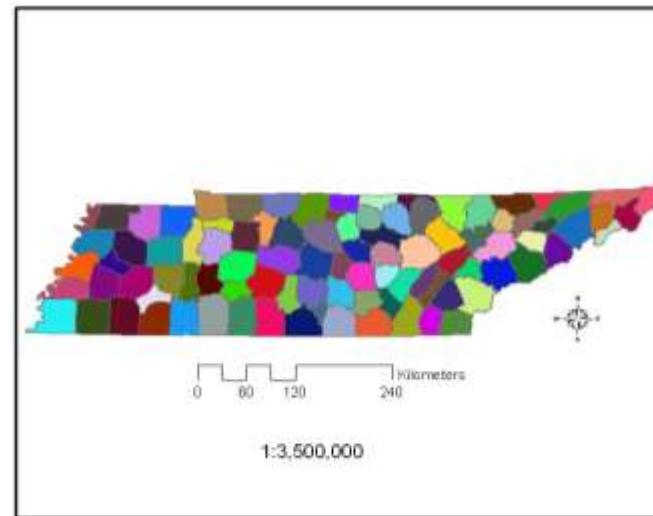
**Map Scale:** The scale is the ratio of distances on the map to the same distance on the ground. It is generally expressed as 1: 100,000, that is 1 cm on the map equals to 100,000 cm on the Earth.

Large scale (ratio is a large fraction) shows small areas with many details

Small scale (ratio is a small fraction) shows larger areas with fewer details



**Large Scale**



**small Scale**

### **Map Scale: Small vs. Large**

Small scale refers to the RF ratio. A 1:250,000 scale is small compared to a 1:2,000. The ratio is small and the amount of reduction is large, producing a map of a large area.

Large scale means less reduction and a map covering a small area.

Scale is expressed in three primary ways:

## 1. Verbal Scale

Map scale is expressed as ordinary text words

ex: 1 centimeter equals (represents) 1 meter .

## 2. Representative fraction (RF)

Map scale is expressed as a ratio in the same units.

ex: 1:2,000 means that one inch (or one meter) on the map represents 2,000 inches (or meters) on the ground.

## 3. Graphic Bar

The graphic bar places visual measure of ground distances on the map.

Used on printed maps (output of GIS) to aid in communicating the scale.

Most software can automatically generate a graphic scale.



## **2. Legend**

The symbol key on a map used to describe a map's symbols and how they are interpreted.

## **Legend**

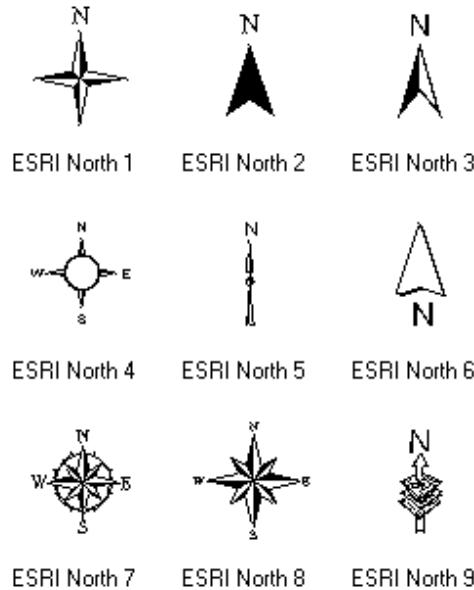
-  Donut
-  Railroad
-  Street

## **RGB Composite**

-  Red: Band\_1
-  Green: Band\_2
-  Blue: Band\_3

## **3. Direction**

**North** is a noun, adjective, or adverb indicating **direction** or geography.



## **4. Sources of information and how processed**

Unless it is absolutely clear from the context in which a map appears, readers will need to know about the sources from which the map was derived. Often the age, accuracy, and reliability of sources is critical to the interpretation of a map and should be noted.

## **5. Title**

The title of a map is usually one of its most essential features. As such, it should receive very careful attention so as to match the needs of the theme and audience. The content of the title should also be measured against other lettering applied to the map, for example in the legend or annotations

## **6. Projection**

The projection used to create a map influences the representation of area, distance, direction, and shape. It should be noted when these characteristics are of prime importance to the interpretation of the map.

**7. Cartographer** The authority lying behind the composition of a map can be of prime importance in some situations. Most maps note the name, initials, or corporate identity of the cartographer(s).

## **8. Date of production**

The meaning and value of some maps--such as those relating to current affairs or weather--are time sensitive. The reader must know when they were produced to estimate whether to trust them or not. An out-of-date road atlas or city map can cause tremendous frustration. Other maps are less sensitive to the passage of time, but the date of production can still be important if, for example, better information becomes available in the period after publication.

## **9. Neatlines**

Neatlines or clipping lines are used to frame a map and to indicate exactly where the area of a map begins and ends. The outer neatline of a map--its border--helps to frame the entire map composition to draw the reader's attention to the various elements of information. Neatlines are also used to "clip" the area of the body of the map and of locator, and inset maps.

## **10. Locator maps**

Some maps portray areas whose locations may be unfamiliar to readers. In such cases, the cartographer adds a "helper" or locator map that places the body of the map within a larger geographical context with which the reader can be expected to be familiar.

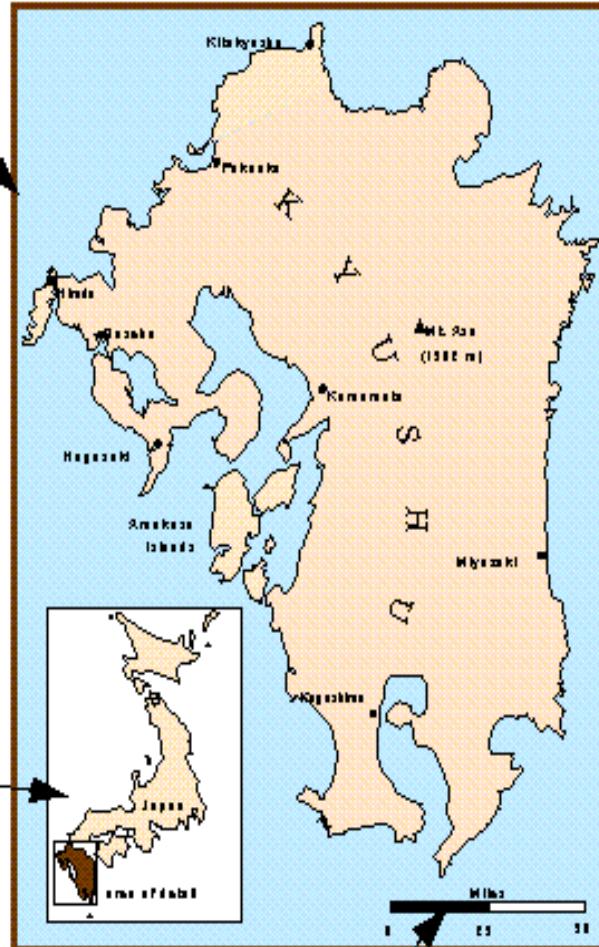
## **11. Inset maps**

Sometimes observations and data are so densely clustered in small sections of a larger map that the cartographer must provide the reader with additional close-up, "zoomed-in" maps of these smaller areas. Otherwise the data will obscure itself. These close-up detailed maps are called insets.

title

# Kyushu and Nagasaki Harbor

neatline



locator map

inset map



Sources: this is where the bibliographic information would go, as well as the projection and cartographer.

scale

bibliographic and production data

## 1.8. Projections and Coordinate Systems

As GIS is based on geographic location you need to understand basics of **Coordinate Systems:**

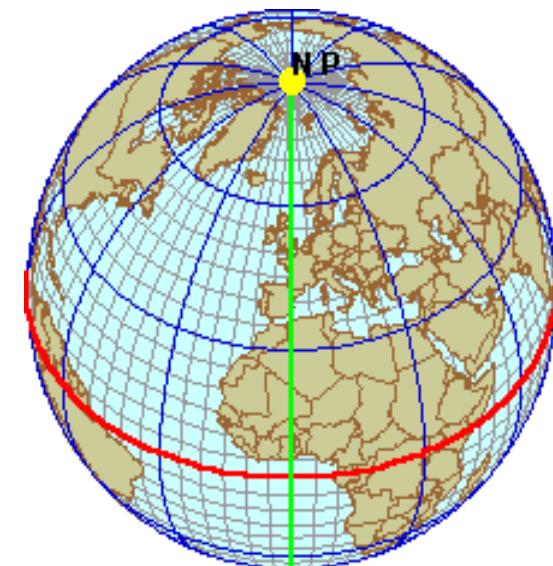
### Geographic Coordinate Systems (GCS)

Location measured from curved surface of the earth

Measurement units latitude and longitude

Degrees-minutes-seconds (DMS)

Decimal degrees (DD) or radians (rad)

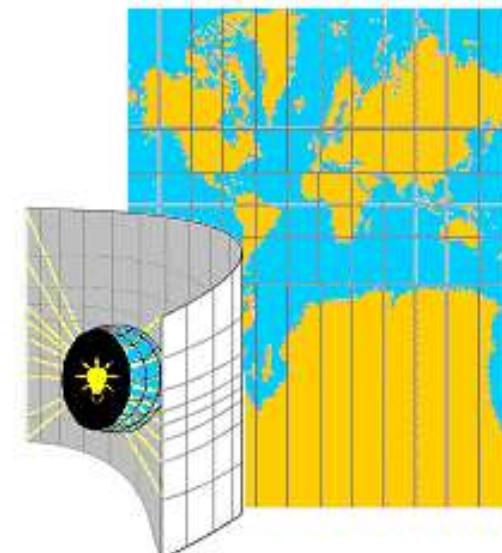


### Projected Coordinate Systems (PCS)

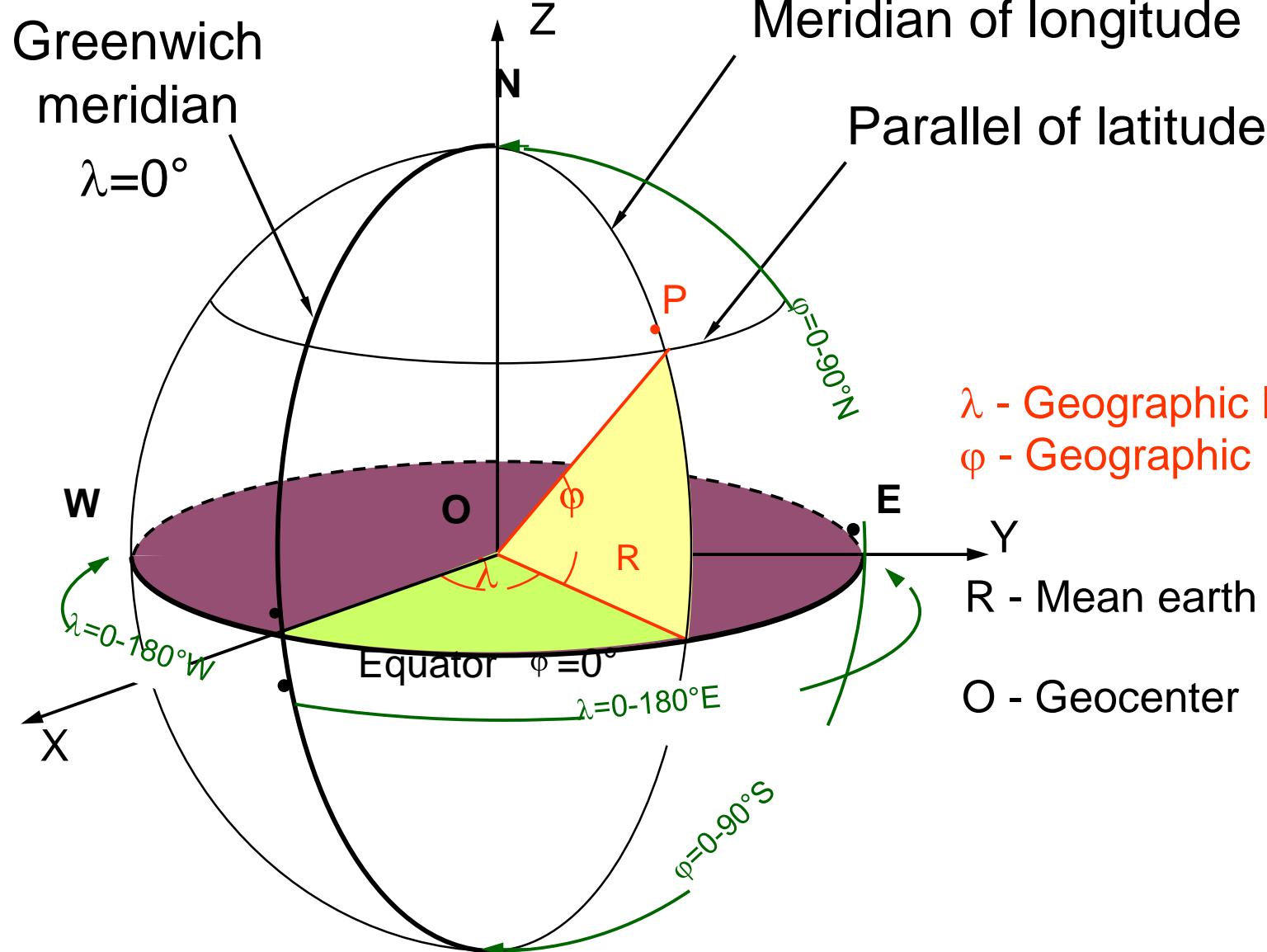
Flat surface

Units can be in meters, feet, inches

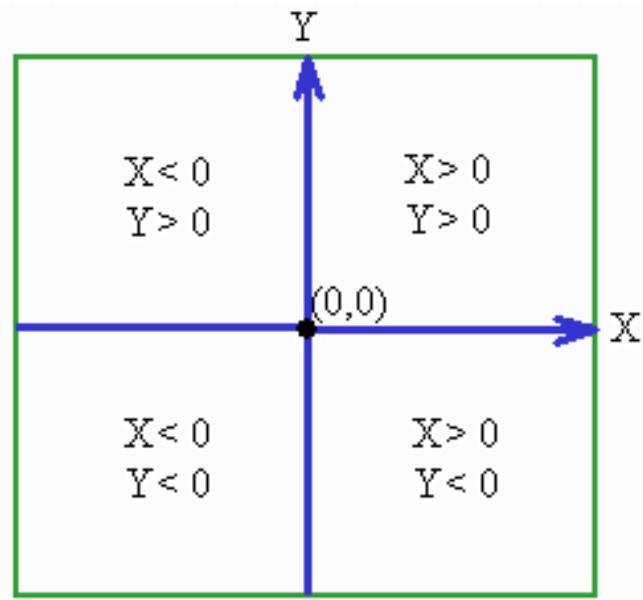
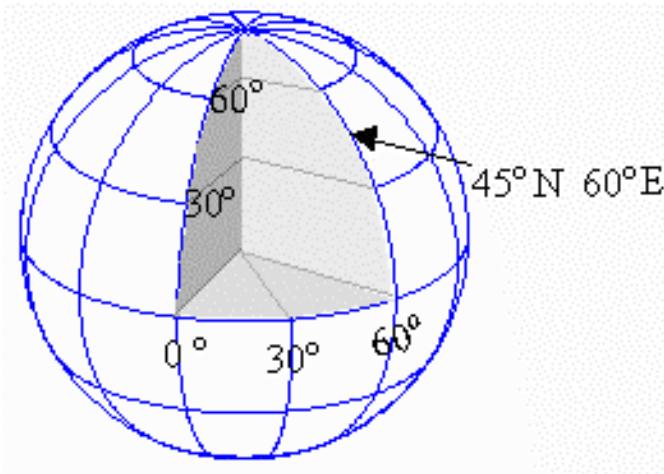
Distortions will occur, except for very fine scale maps



## 1.8.1. Geographic Coordinate Systems (GCS)



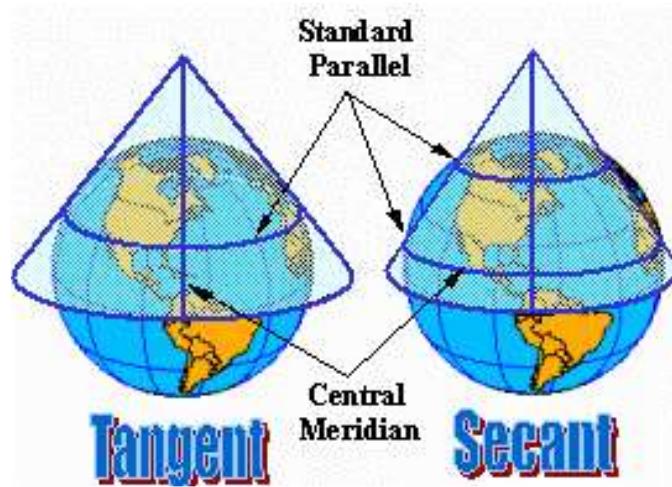
# Geographic and Projected Coordinates



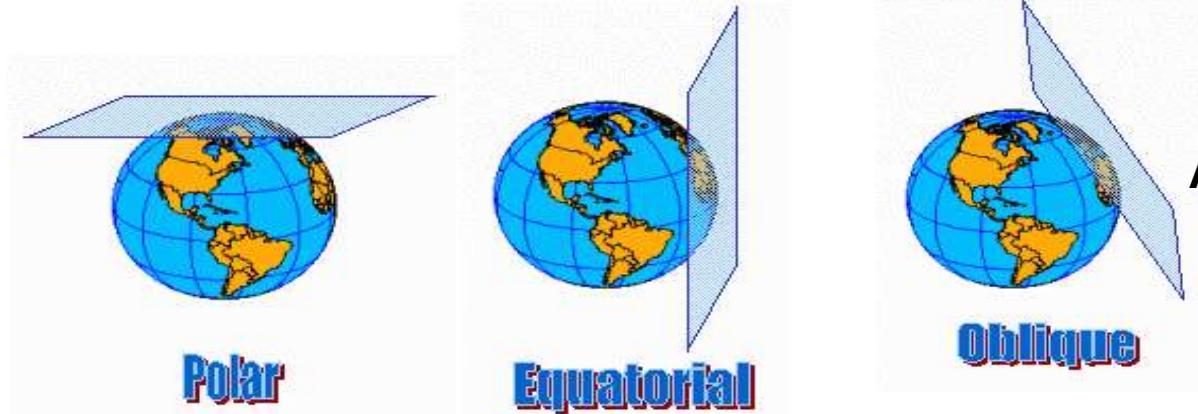
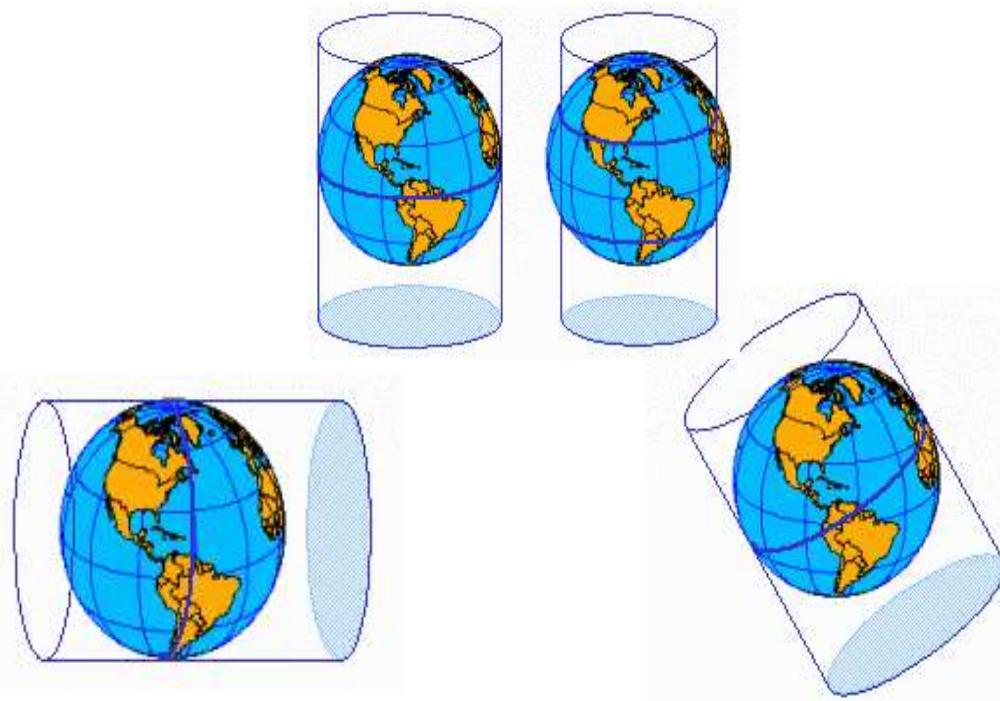
$(\phi, \lambda)$   $\longleftrightarrow$   $(x, y)$   
Map Projection

## 1.8.2. Projected Coordinate Systems (PCS)

### Cylindrical Projections



### Conic Projections



### Azimuthal Projections

Oblique



Cairo University



# Geographic Information System And Spatial Databases

Course Code: IS443

## Chapter 2: Spatial Databases

# **Chapter 2: Spatial Databases**

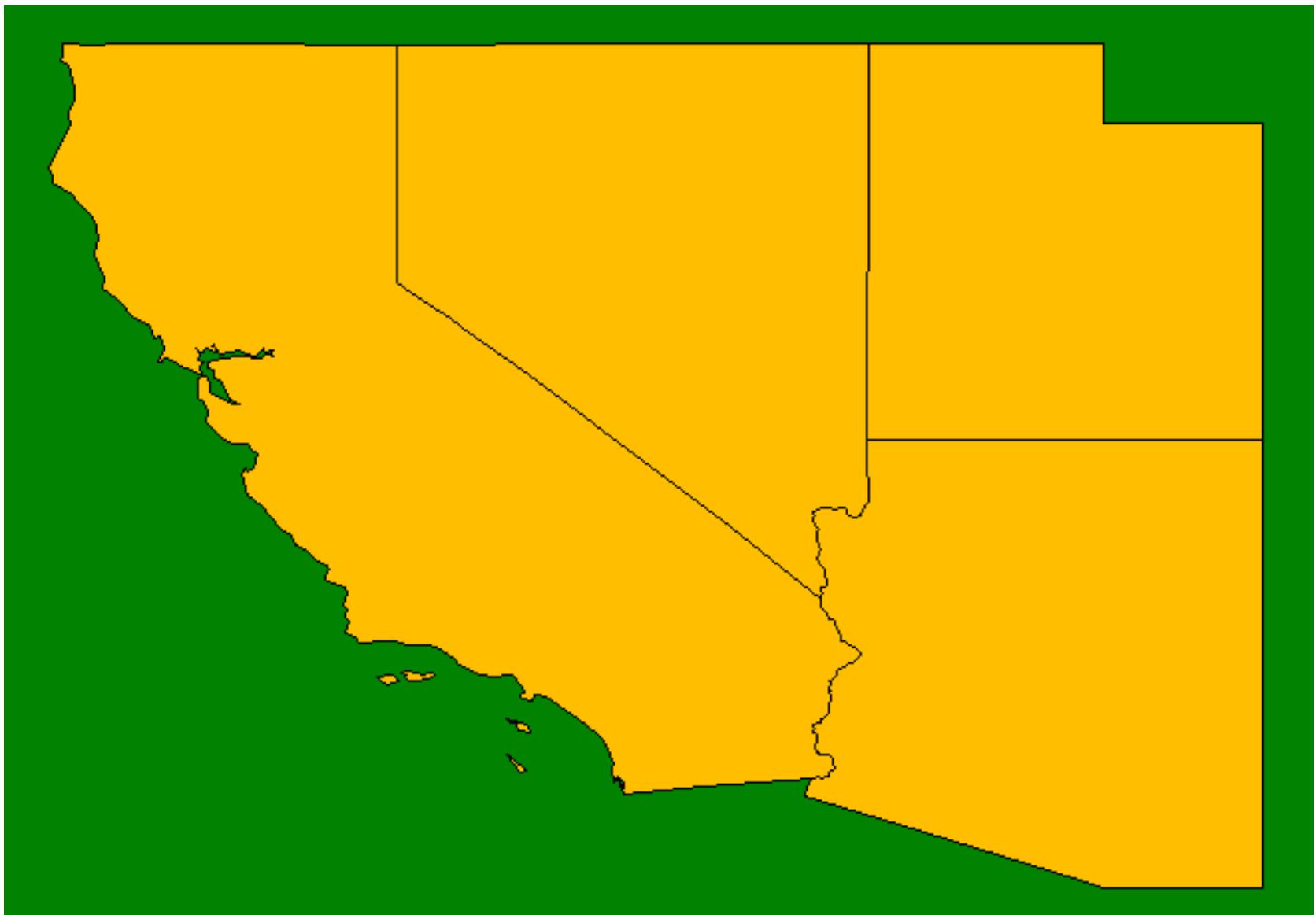
## **2.1. Vector Data Model**

## Terminology

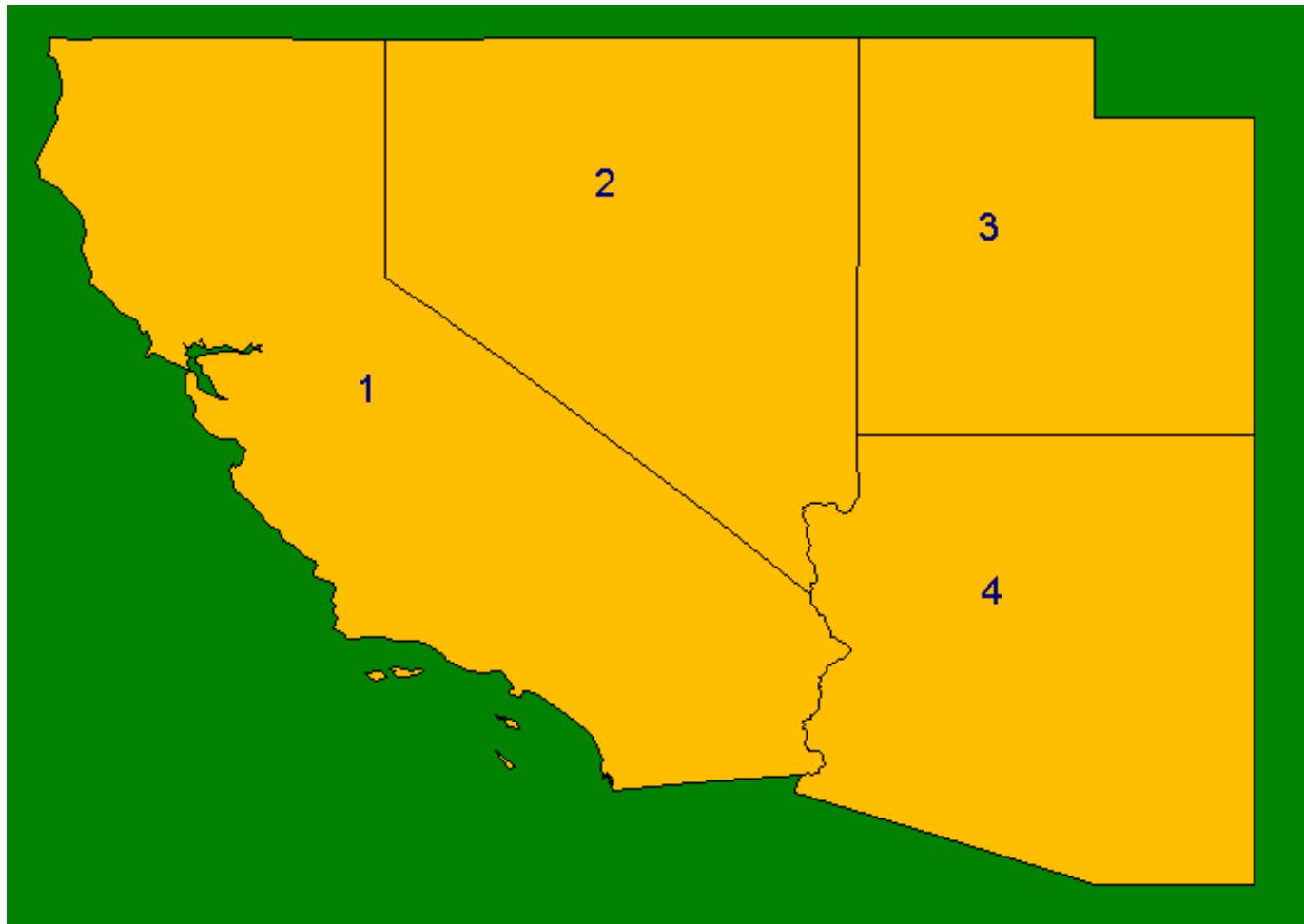
- **Point**: x, y coordinate identifying a geographic location
- **Node**: the beginning and end of link (often defined where 3 or more lines connect)
- **VERTICES** define the shape of the line and occupy the location between the two nodes



- **Link** (line, arc): an ordered set of points with a node at the beginning and end of it
- **Polygon**: two or more links connected at the nodes, contains a point inside to identify the polygons attributes



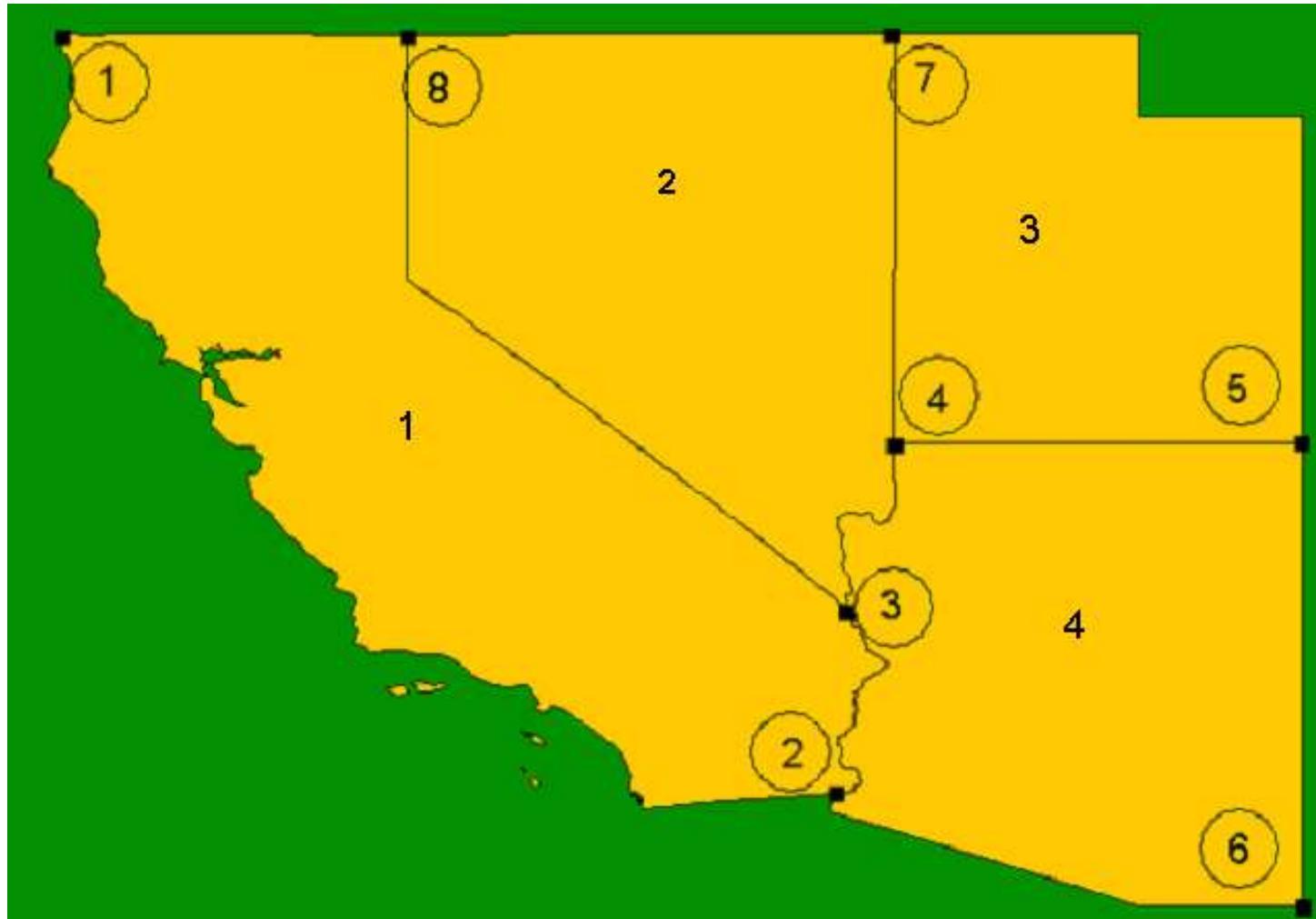
# Identify the polygons



# Create the polygon attribute table (PAT)

Poly-ID	Name	Population
1	California	33090214
2	Nevada	1818259
3	Utah	2135252
4	Arizona	4790311

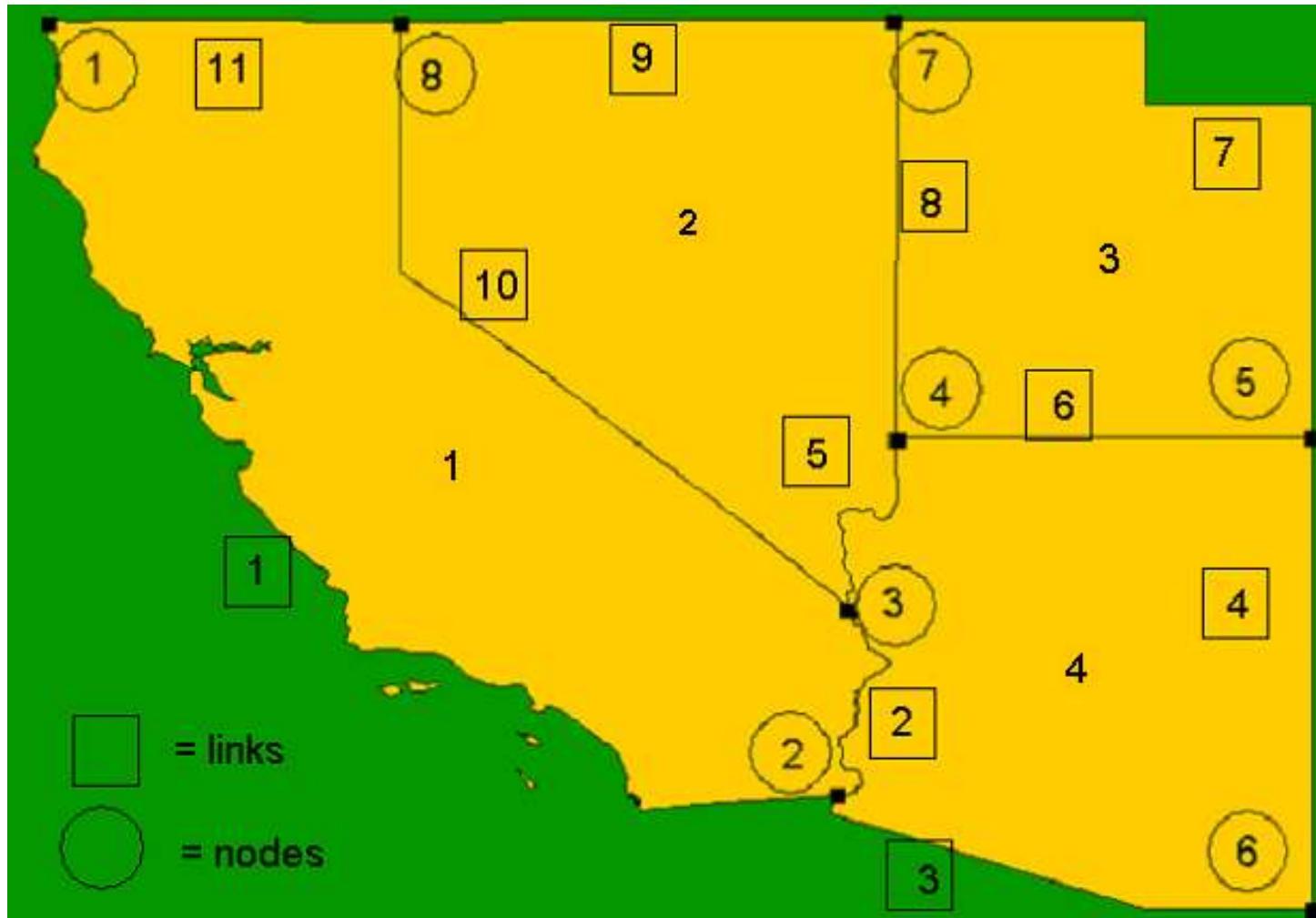
# Identify the nodes



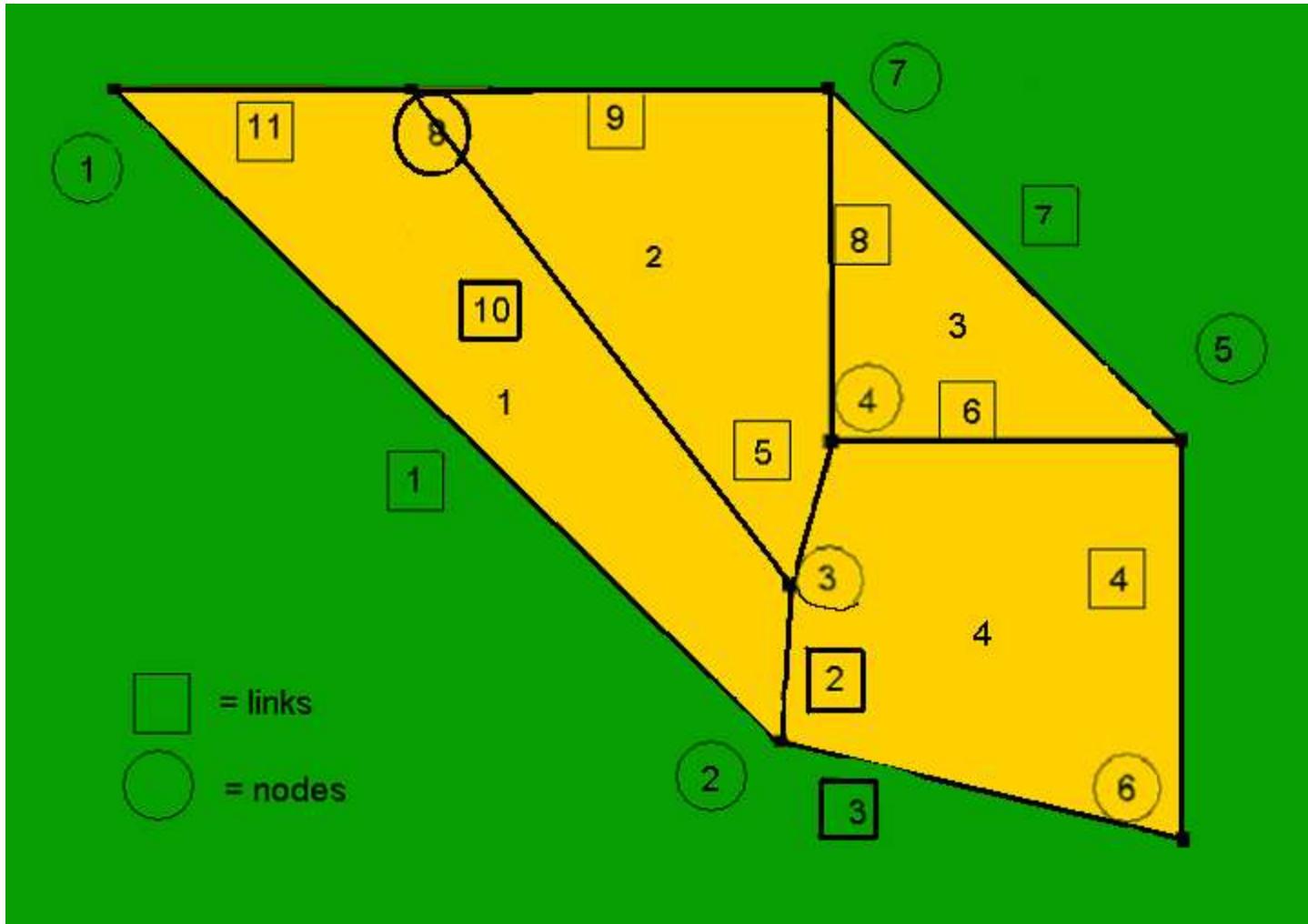
# Node table

Node ID	X-coord	Y-coord
1		
2		
3		
4		
5		
6		
7		
8		

# Identify the links (arcs, lines)



# Simplify this



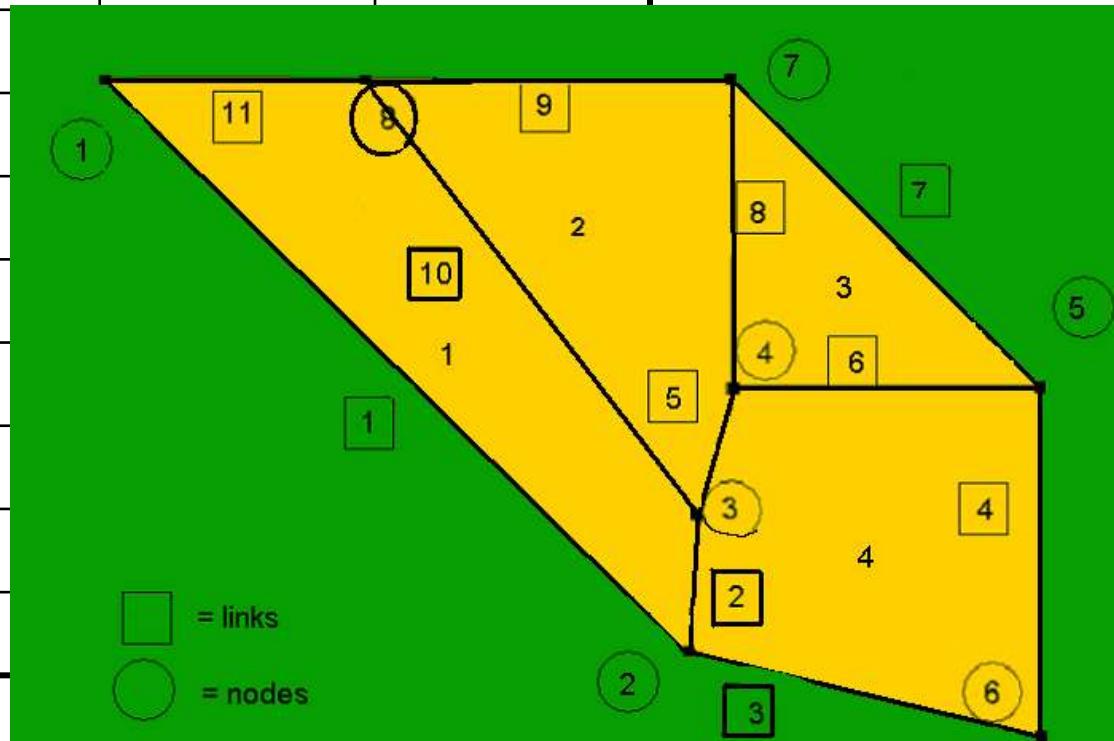
# Create the topology!

## Links table

Link#	FNode	TNode	LPoly	RPoly
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				

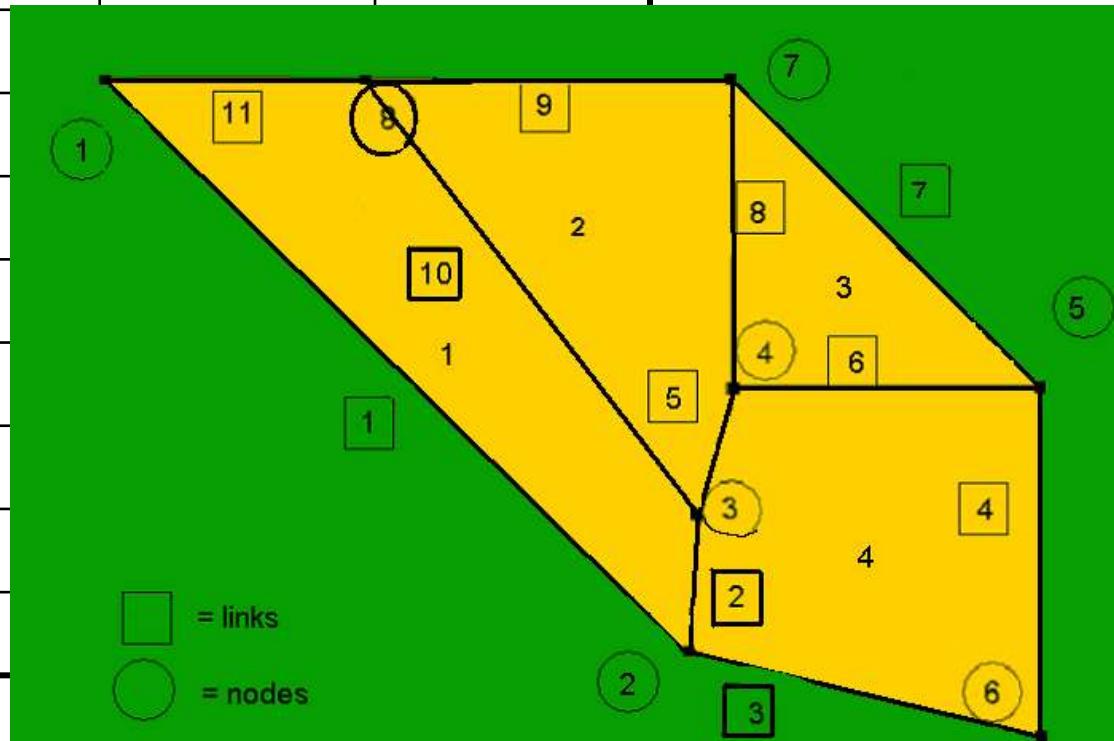
# Nodes First

Link#	FNode	TNode	LPoly	RPoly
1	1	2		
2	2	3		
3	2	6		
4	6	5		
5	3	4		
6	4			
7	5			
8	4			
9	7			
10	3			
11	8			



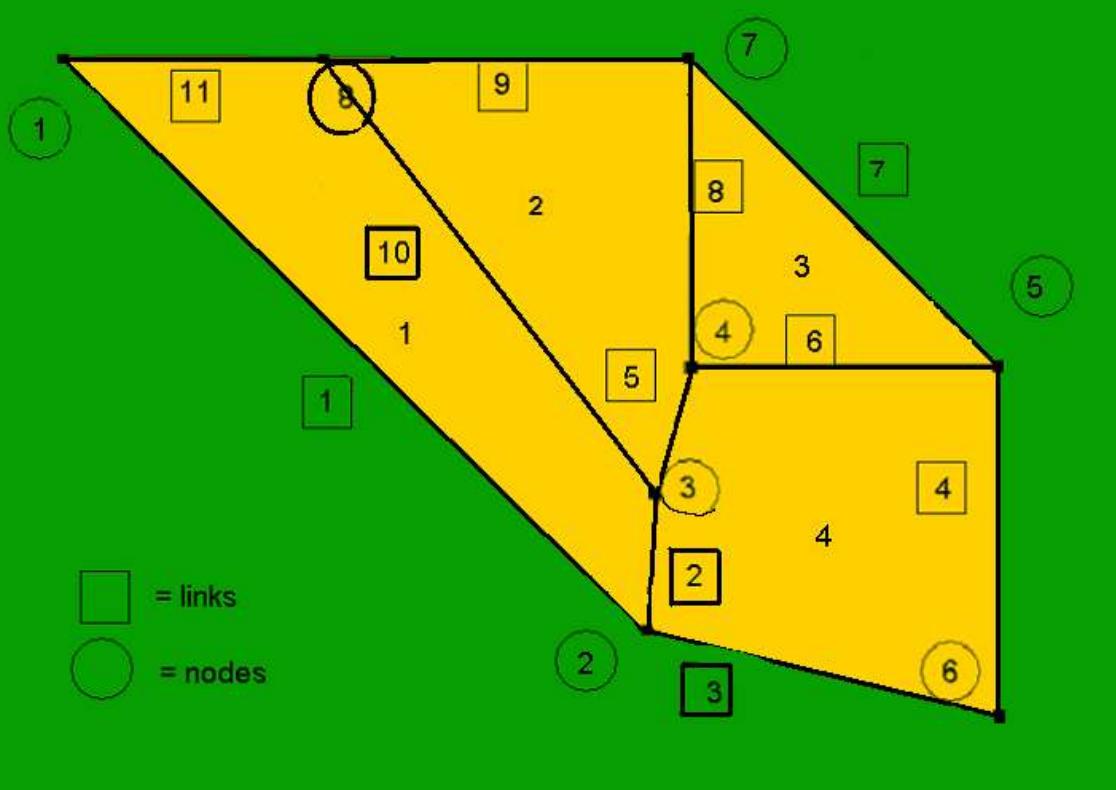
# Nodes First

Link#	FNode	TNode	LPoly	RPoly
1	1	2		
2	2	3		
3	2	6		
4	6	5		
5	3	4		
6	4	5		
7	5	7		
8	4	7		
9	7	8		
10	3	8		
11	8	1		



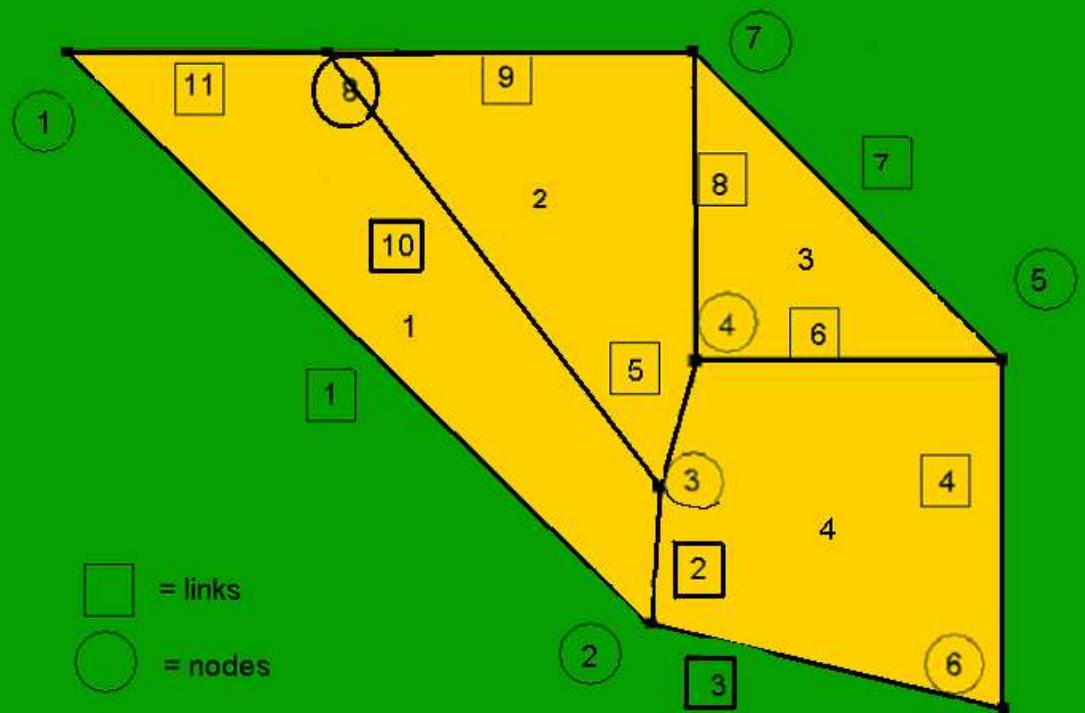
# Polygons

Link#	FNode	TNode	LPoly	RPoly
1	1	2	1	0
2	2	3	1	4
3	2	6	4	0
4	6	5	4	0



# Polygons

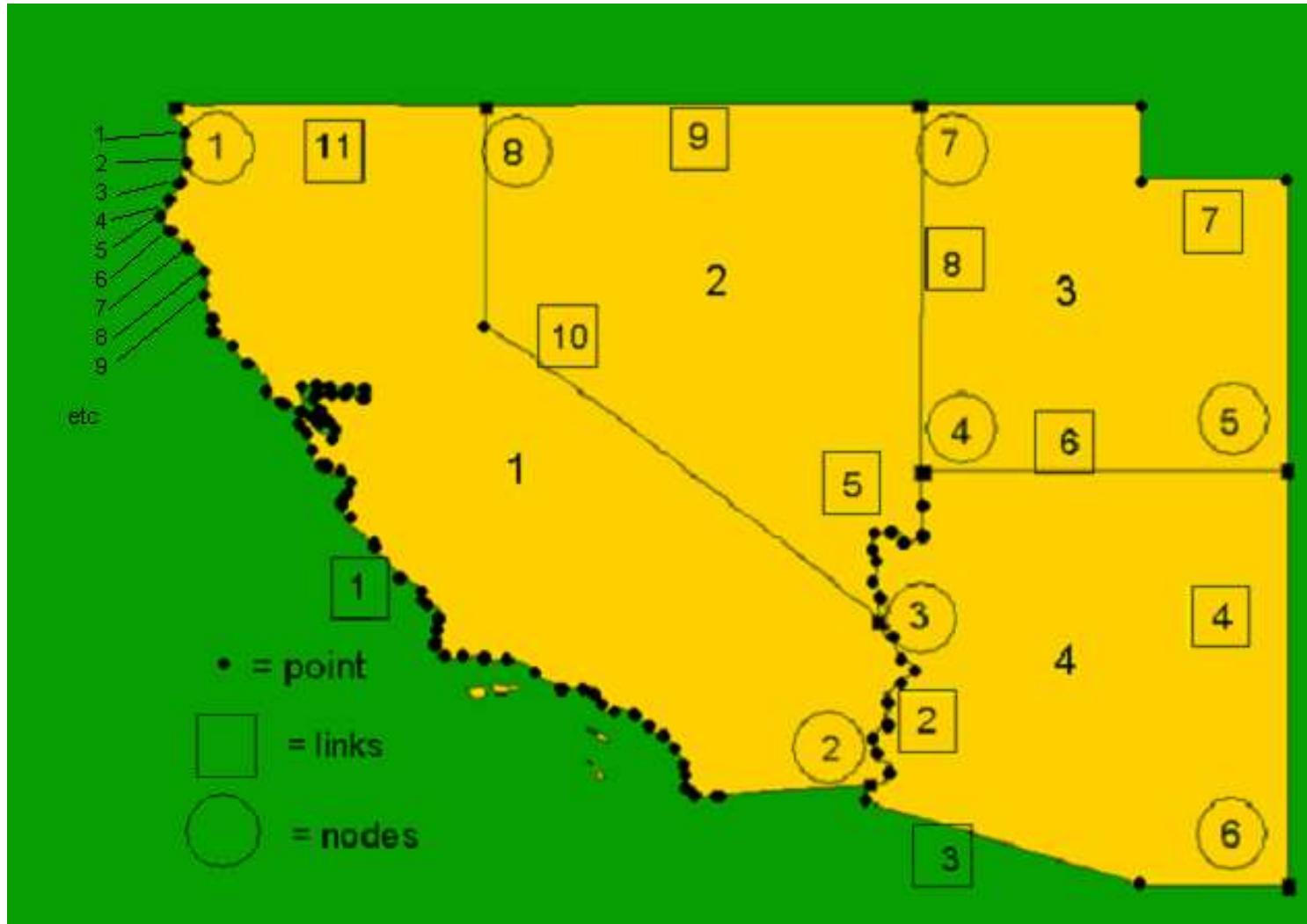
Link#	FNode	TNode	LPoly	RPoly
1	1	2	1	0
2	2	3	1	4
3	2	6	4	0
4	6	5	4	0
			2	4
			3	4
			3	0
			2	3
			2	0
			1	2
			1	0



□ = links

○ = nodes

# Identify the points



# Link List

Link#	List of points
1	1,2,3,4,5,6,7,8,9,10... etc
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	

# Point Coordinates

ID	X-coord	Y-coord
1		
2		
3		
4		
5		
6		
7		
8		
9 (etc)		

# ting it all together

Poly-ID	Name	Population
1	California	33090214
2	Nevada	
3	Utah	
4	..	
Link#	List of points	
1	1,2,3,4,5,6,7,8,9,10..	
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		

Point-ID	X-coord	Y-coord		
1				
2				
Link#	FNode	TNode	LPoly	RPoly
1	1	2	1	0
2	2	3	1	4
3	2	6	4	0
4	6	5	4	0
5	3	4	-	-
6	4	5		
7	5	7		
8	4	7		
9	7	8		
10	3	8		
11	8	1		

Node ID	X-coord	Y-coord
1		
2		
3		
4		
5		
6		
7		
8		

# Putting it all together

Poly-ID	Name	Population	Point-ID	X-coord	Y-coord
1	California	33090214	1		
2	Nevada		2		
3	Utah				
4	Arizona				
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					
6					
7					
8					
9					
10					
Link#	List of points				
1	1,2,3,4,5,6,7,8,9,10				
2					
3					
4					
5					

# Putting it all together

Poly-ID	Name	Population	Link#	FNode	TNode	LPoly	RPoly
1	California	33090214	1	1	2	1	0
2	Nevada		2	2	3	1	4
3	Utah		3	2	6	4	0
4	Arizona		4	6	5	4	0
Link#	List of points		5	3	4		
1	1,2,3,4,5,6,7,8,9,10...		6	4	5		
2			7	5	7		
3			8	4	7		
4			9	7	8		
5			10	3	8		
6			11	8	1		
Node ID	X-coord	Y-coord					
1							
2							
3							
4							
5							
6							
7							
8							

# Putting it all together

Poly-ID	Name	Population
1	California	33090214
2	Nevada	
3	Utah	
4	Arizona	

Point-ID	X-coord	Y-coord
1		
2		
LPoly	RPoly	
1	0	
2	4	
4	0	
0		

Link#	List of points
1	1,2,3,4,5,6,7,8,9,10
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	

Node ID	X-coord	Y-coord
1		
2		
3		
4		
5		
6		
7		
8		

# Putting it all together

Poly-ID	Name	Population	Point-ID	X-coord	Y-coord
1	California	33090214	1		
2	Nevada		2		
3	Utah				
4	Arizona				
Link#	List of points	FNode	TNode	LPoly	RPoly
1	1,2,3,4,5,6,7,8,9,10	1	2	1	0
2		2	3	1	4
3		2	6	4	0
4		6	5	4	0
Link#	List of points	5	4	1	0
5		3	4		
6		4	5		
7		5	7		
8		4	7		
9		7	8		
10		3	8		
11		8	1		
Node ID	X-coord	Y-coord	Node ID	X-coord	Y-coord
1			1		
2			2		
3			3		
4			4		
5			5		
6			6		
7			7		
8			8		

# Spatial relationships

- “adjacent to”
- “connected to”
- “near to”
- “intersects with”
- “within”
- “overlaps”
- etc.

# Spatial relationships

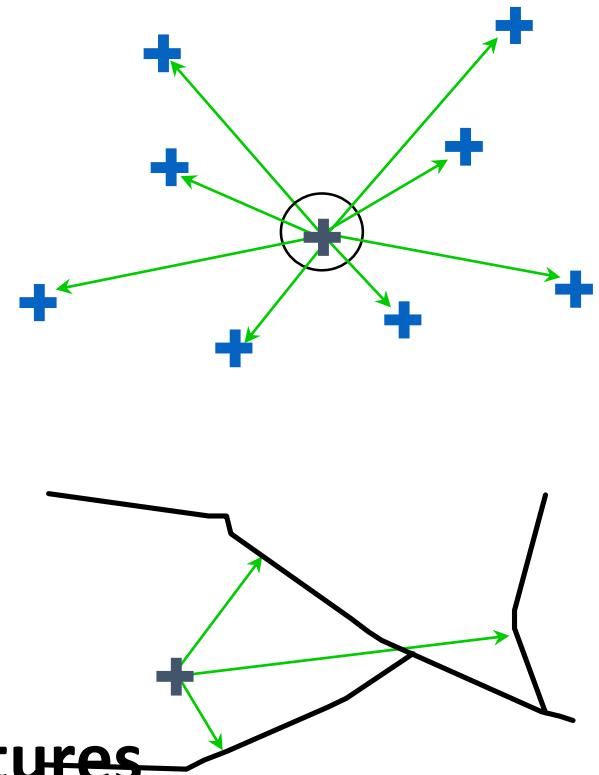
- **logical connections between spatial objects represented by points, lines and polygons**
- **e.g.,**
  - **point-in-polygon**
  - **line-line**
  - **polygon-polygon**

# Spatial relationships

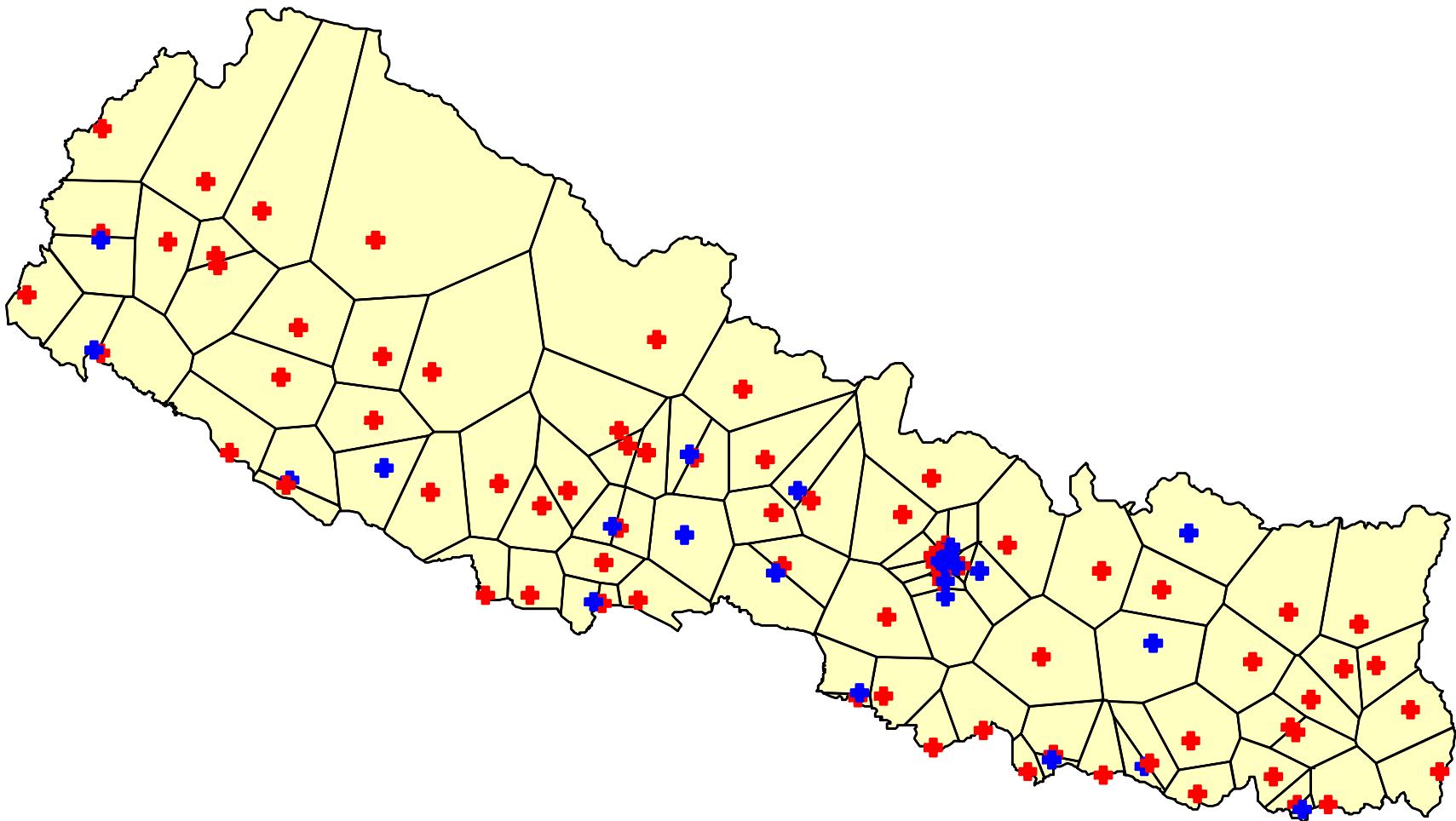
- some relationships are stored in a topological data model
  - “adjacent to”
    - right poly and left poly in the line attribute table
  - “connected to”
    - list of lines that share the same node in the node attribute table
- others need to be computed

# “is nearest to”

- point/point
  - which family planning clinic is closest to the village?
- point/line
  - which road is nearest to the village
- same with other combinations of spatial features

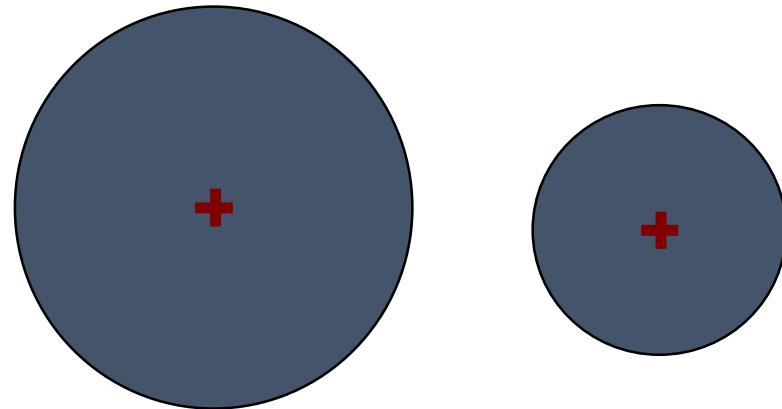


# “is nearest to”: Thiessen polygons



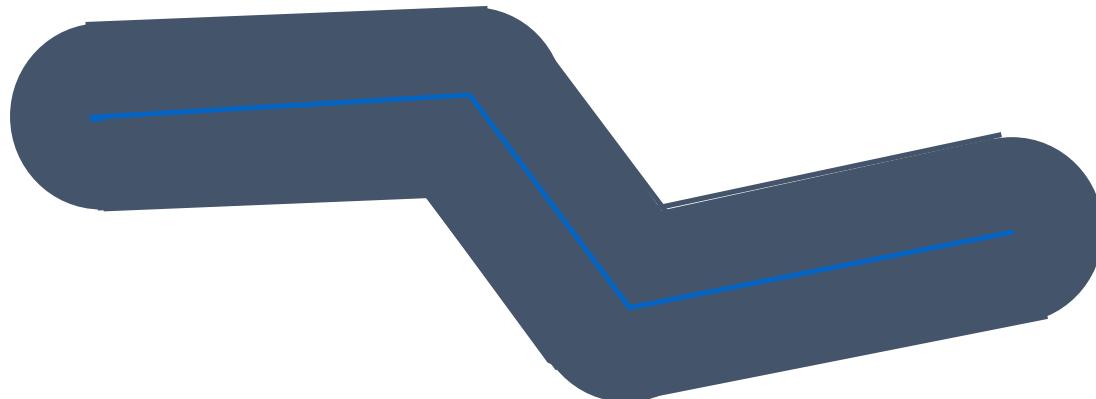
# “is near to”: buffer operations

- **point buffer**
  - affected area around a polluting facility
  - catchment area of a water source



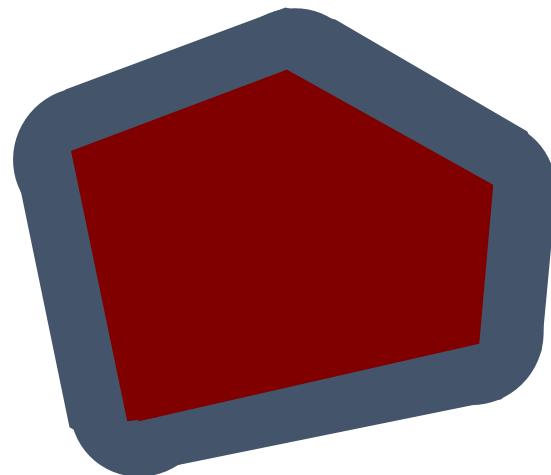
# buffer operations

- **line buffer**
  - how many people live near the polluted river?
  - what is the area impacted by highway noise?



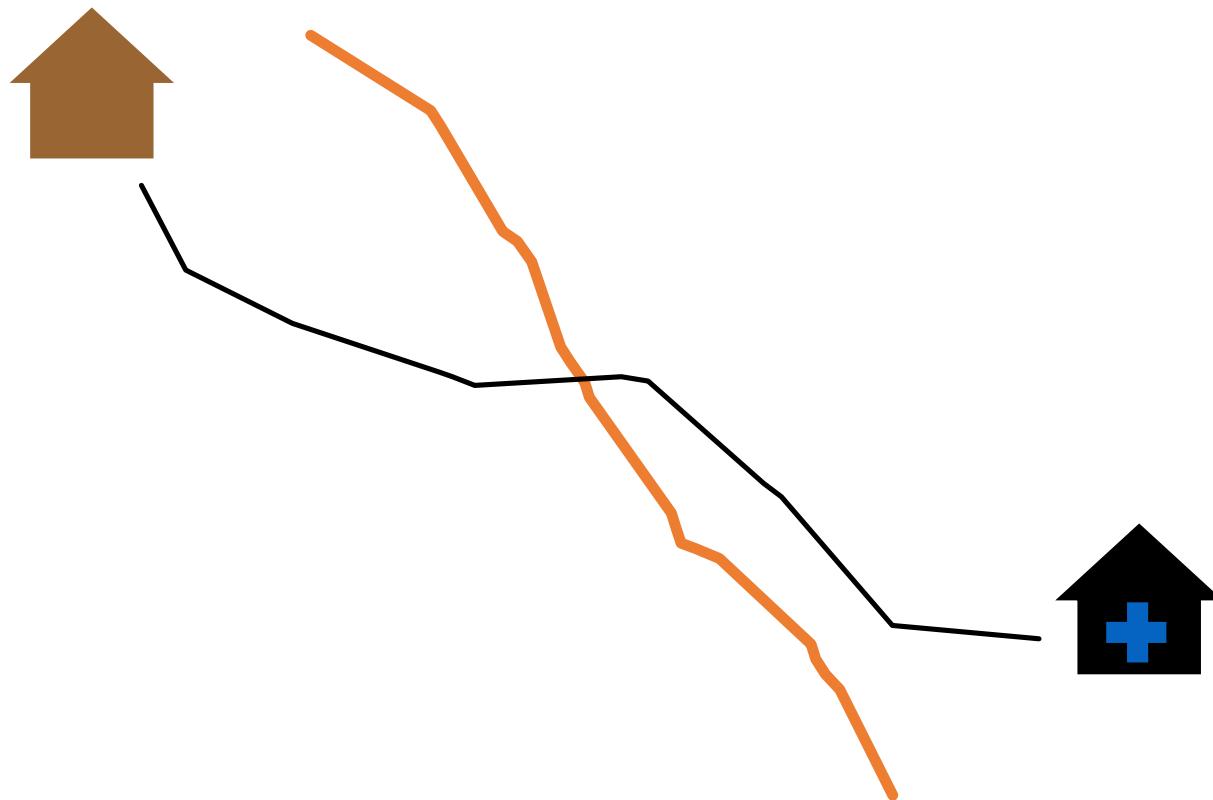
# buffer operations

- **polygon buffer**
  - area around a reservoir where development should not be permitted



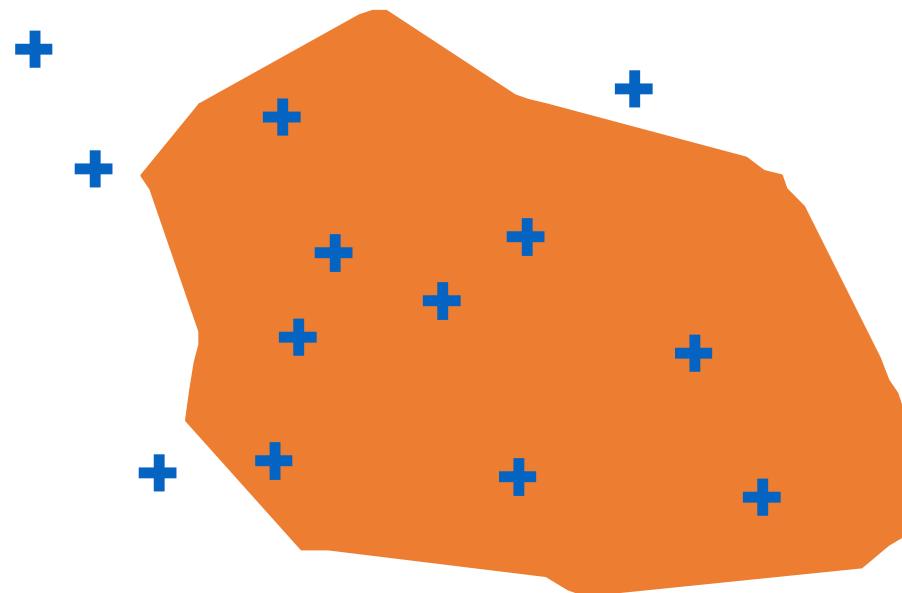
# “crosses”: line intersection

- when traveling to the dispensary, do farmers have to cross this river?

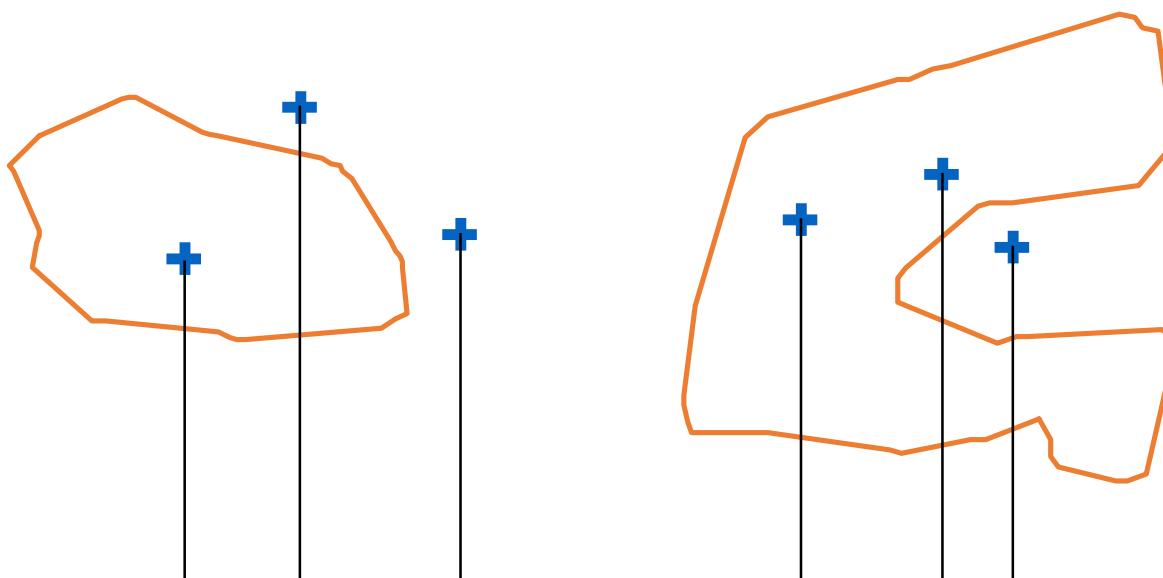


# “is within”: point in polygon

- which of the cholera cases are within the contaminated water catchment area?



# Point in polygon



even number of intersections: point is outside  
odd number of intersections: point is inside

# TOPOLOGY (for vector data)

- What is topology?
- Why is important?
- Three types of topological models in GIS
- Spatial operations of topology
  - Contiguity
  - Connectivity
- Trade-offs of topological structure

# Spatial features and spatial relationships

- Spatial features in maps
  - Points, lines and polygons
- Human being interprets additional information from maps about **the spatial relationships** between features
  - A route trace from an airport to a house
  - Land contiguity adjacent to streets along which the lands are located

# The definition of Topology

- The spatial relationships can be interpreted
  - identification of connecting lines along a path
  - definition of the areas enclosed within these lines
  - identification of contiguous areas
- In digital maps, these relationships are depicted using '**Topology**'
- Topology = *A mathematical procedure for explicitly defining spatial relationship*
- Topology is the description of how the spatial objects are related with spatial meaning

# Topological data models

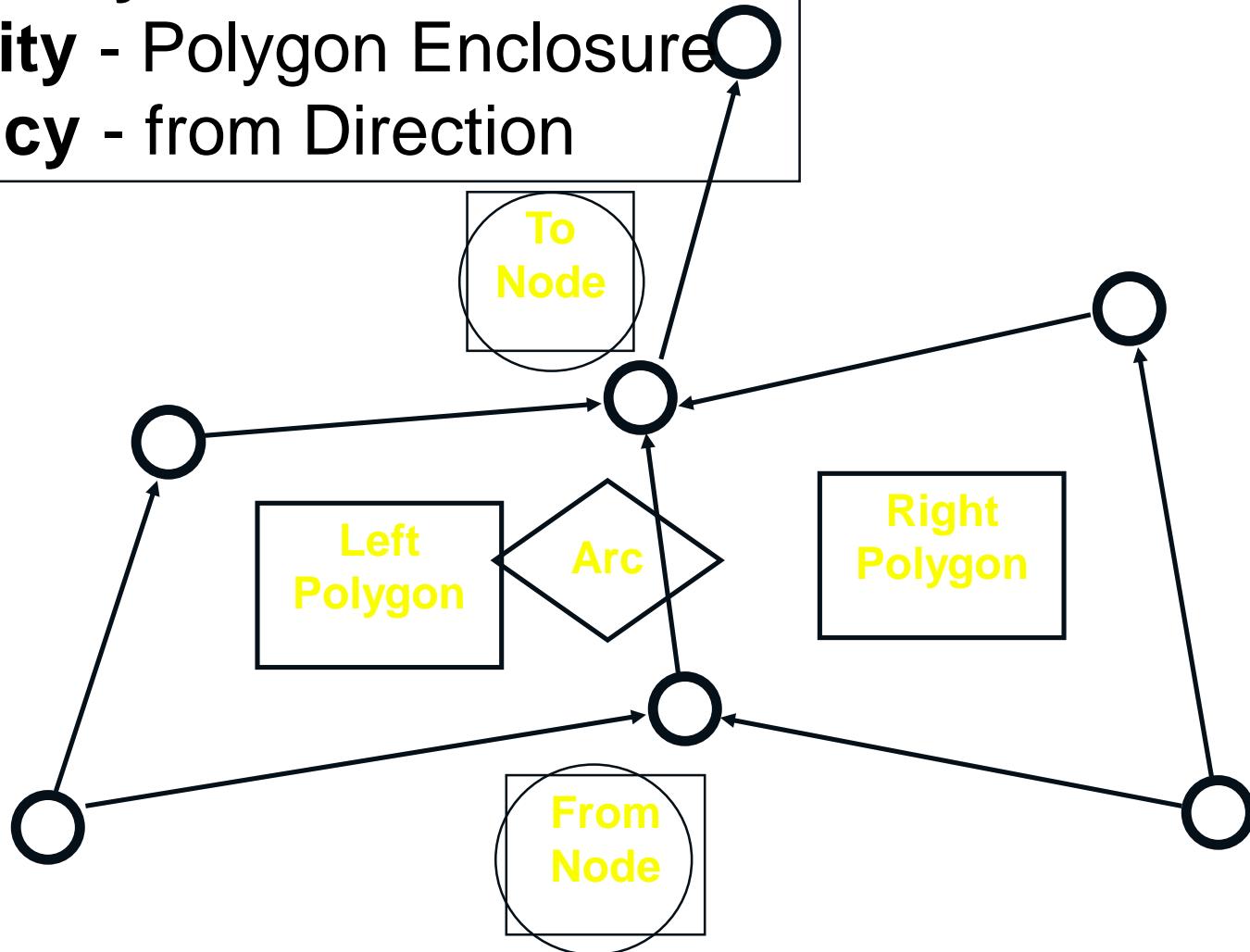
- Three types of topological concepts
  - Arc, Node and polygon topologies
- Arc
  - Arcs have directions and left and right polygons (=contiguity)
- Node
  - Nodes link arcs with start and end nodes (=connectivity)
- Polygon
  - Arcs that connect to surround an area define a polygon (=area definition)

# Terms and concepts

**Connectivity** - from and to nodes

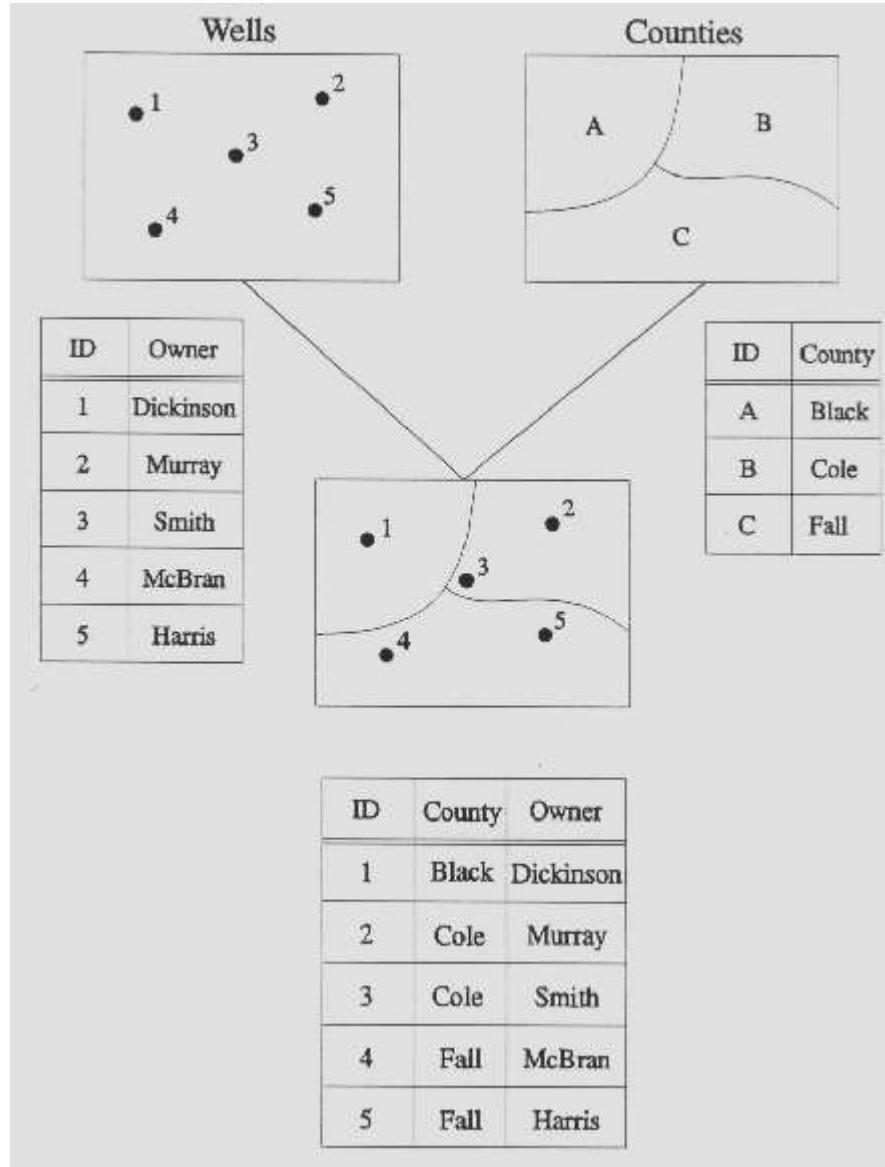
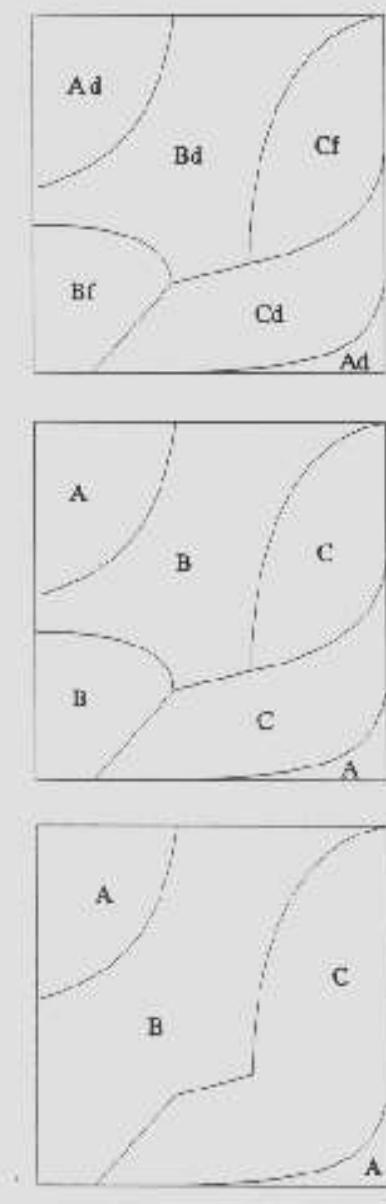
**Contiguity** - Polygon Enclosure

**Adjacency** - from Direction

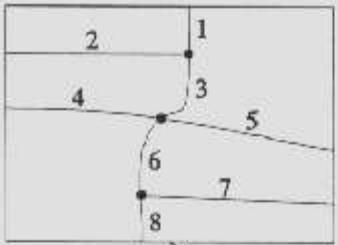


# Spatial operations of topology

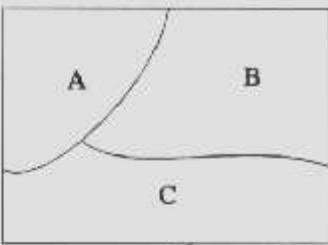
- Connectivity and contiguity (Aronoff, 1989)
  - A basic, but core spatial analysis operations in GIS
- Contiguity
  - A biologist might be interested in the habitats that occur next to each other
  - A city planner might be interested in zoning conflicts such as industrial zones bordering recreation areas
- Connectivity
  - Transportation network, telecommunication systems, river systems
  - To find optimum routings or most efficient delivery routes or the fastest travel route
  - To predict loading at critical points in a river channel
  - To estimate water flow at a bridge crossing that will result from heavy flood



Roads

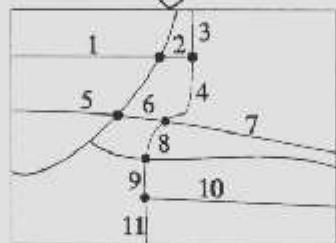


Counties



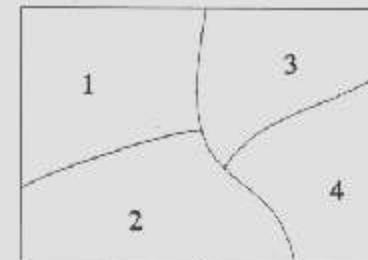
ID	Hwy.
1	35
2	22
3	35
4	60
5	60
6	35
7	82
8	35

ID	County
A	Black
B	Cole
C	Fall

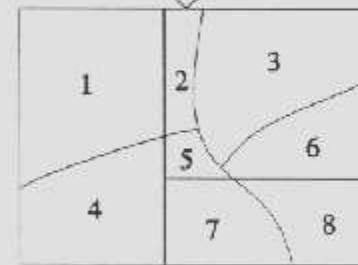
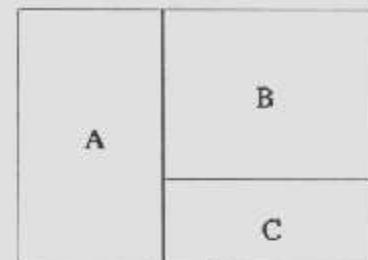


ID	Original	Hwy.	County
1	2	22	Black
2	2	22	Cole
3	1	35	Cole
4	3	35	Cole
5	4	60	Black
6	4	60	Cole
7	5	60	Cole
8	6	35	Cole
9	6	35	Fall
10	7	82	Fall
11	8	35	Fall

Watershed



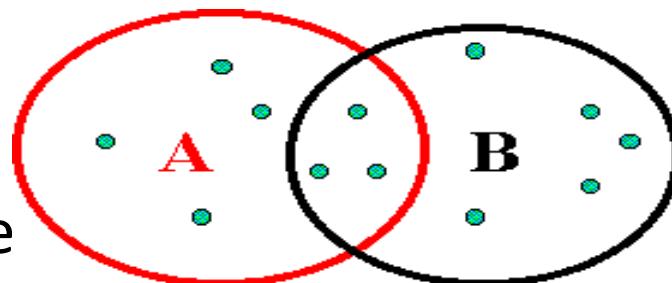
Counties



ID	Watershed ID	County ID
1	1	A
2	1	B
3	3	B
4	2	A
5	2	B
6	4	B
7	2	C
8	4	C

# Overlay operations in a GIS

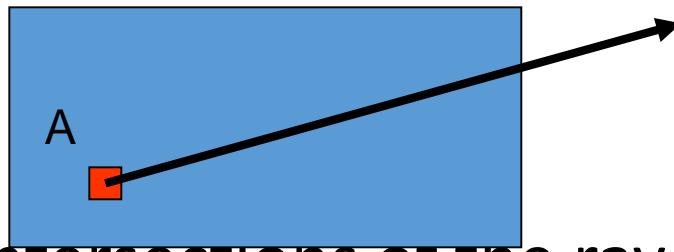
- Origins in Landscape Planning
  - Literally overlaying maps on a light table and searching for overlapping areas
  - GIS started out at GSD in the 1960s
- Set theory – polygons represent sets, overlay represents intersects and unions



- Computational Ge

# Simplest form of overlay

## Point in polygon procedure



Line intersects  
1 edge of polygon

Odd number of  
Intersections = inside,  
Even = outside

- Count how many intersections of the ray, originating at point A, pass through edges of the polygon

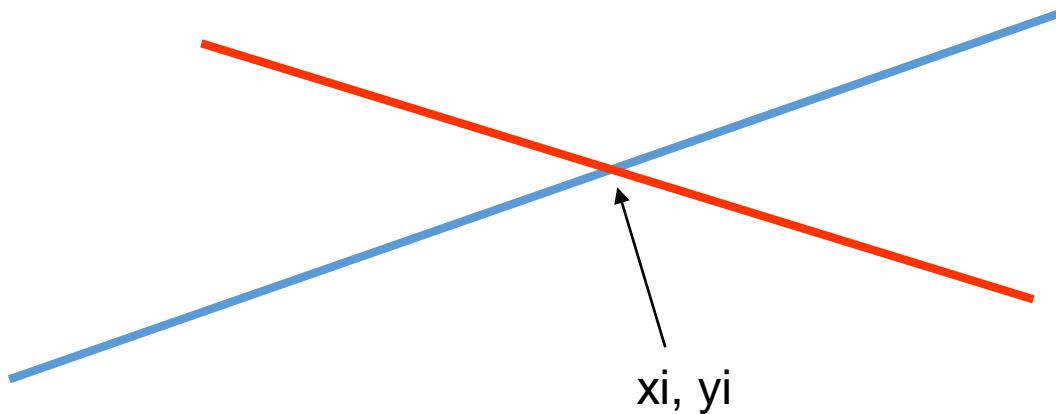
# Line intersections

$y = a_1 + b_1x$  and  $y = a_2 + b_2x$

intersect at:

$$x_i = - (a_1 - a_2) / (b_1 - b_2), y_i = a_1 + b_1 x_i$$

And checking for the values of  $x_i$  to see that  
It falls within the  $x$  values of each of the lines.



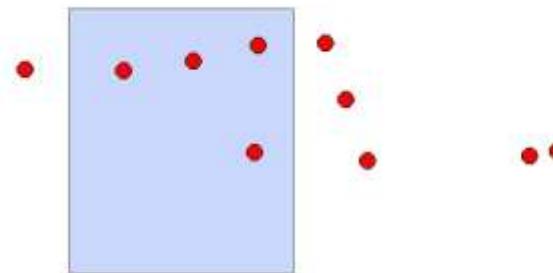
# General overlay types

- Identity
  - spatial join or point-in-polygon
- Clip
  - similar to set extent when using raster data
- Intersection
- Union
- Buffer

(for all of the above, operations are on layers,  
not single polygons)

# Spatial Join

Point in polygon  
operation – which  
points are in the  
Polygon?

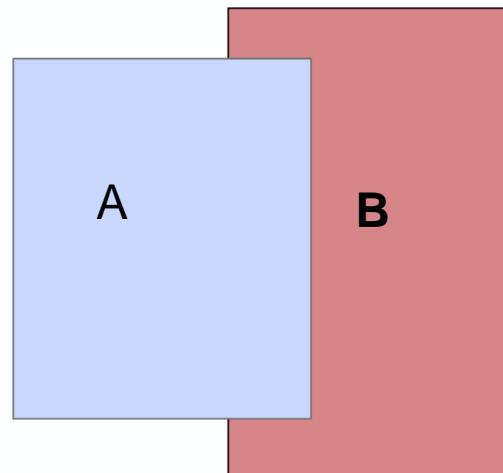


Polygon ID (id\_1) is  
added to the point  
layer's attribute table.

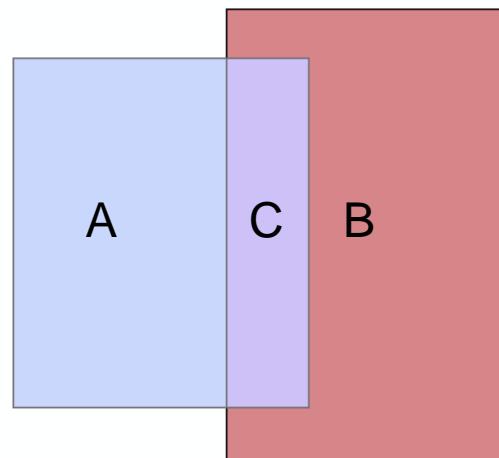
Attributes of Join_Output						
FID	Shape	FID_1	Id	FID_2	Id_1	
0	Point		1	2	0	1
1	Point		2	3	0	1
2	Point		8	9	0	1
3	Point		9	10	0	1
4	Point		0	1	0	0
5	Point		3	4	0	0
6	Point		4	5	0	0
7	Point		5	6	0	0
8	Point		6	7	0	0
9	Point		7	8	0	0

# Clip

Two polygons, A and B, overlap. Clip A using B as a cookie cutter.

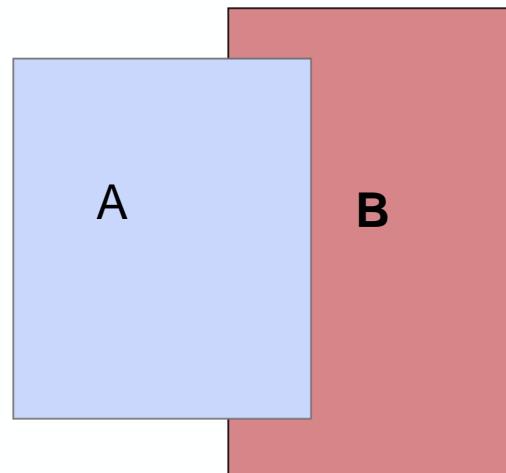


Clip operation creates a new polygon, C, which is the intersect, or overlap, of A and B. Attributes of A do not appear in C.

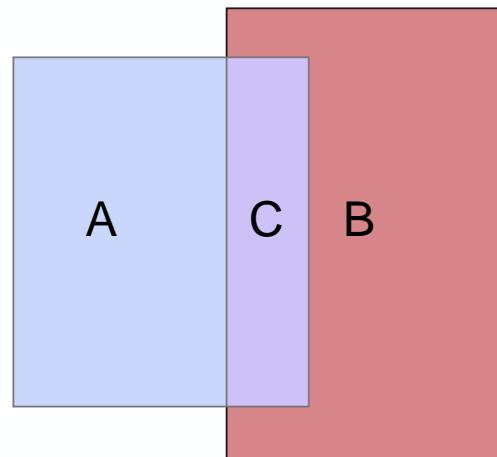


# Intersect

Two polygons, A and B,  
Overlap. Find the  
Intersection of A using B.

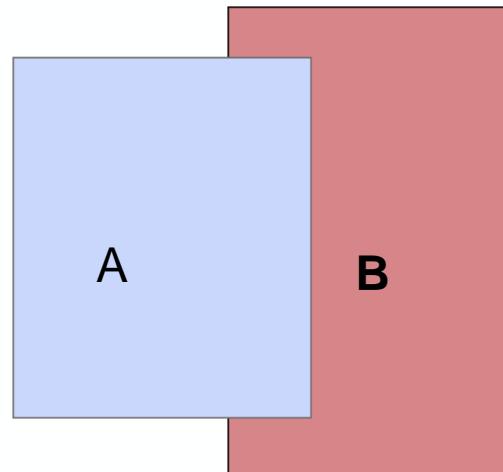


Intersect operation creates a new polygon, C, which is the intersection, or overlap, of A and B. Attributes of A and B do appear in C.

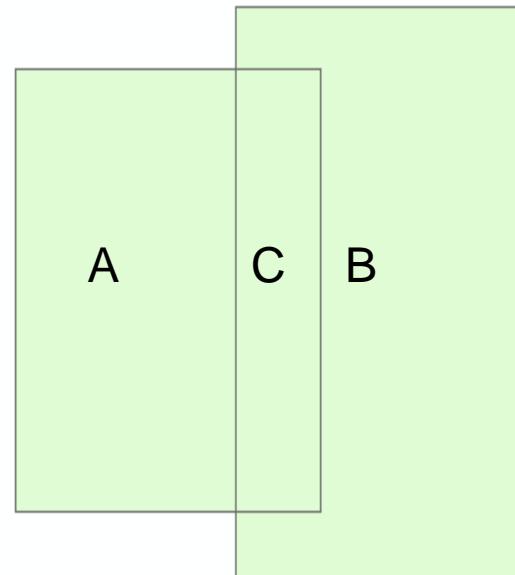


# Union

Two polygons, A and B, Overlap. Find the intersection of A using B.



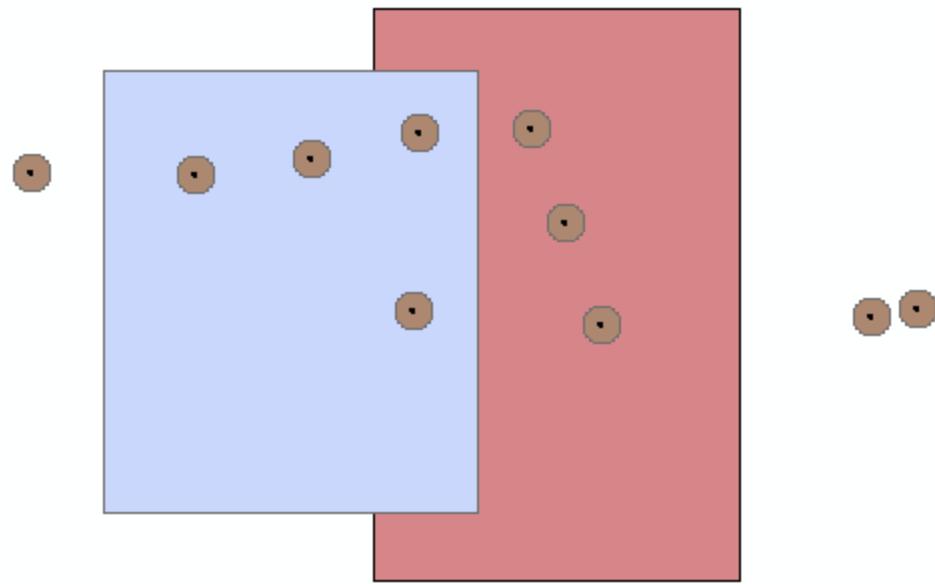
Intersect operation creates a new polygon, C, which is the intersect, or overlap, of A and B. Attributes of A do appear in C. A and B are also part of the union and retain their attributes.



# Buffer

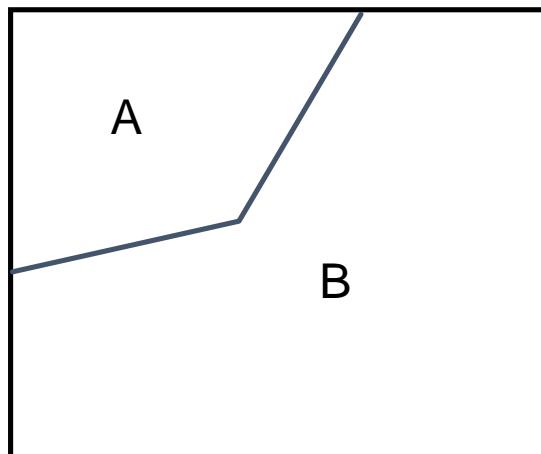
Buffers are polygon shapes that surround a feature by a uniform distance. Buffers can be created around points, lines, and polygons.

Buffers don't share the attributes of the feature that they surround. Use spatial Joins to add the attributes.

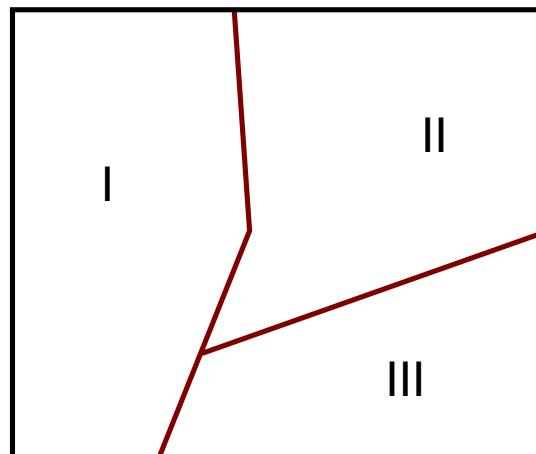


Original points (black) are surrounded by a buffer of 25 meters.

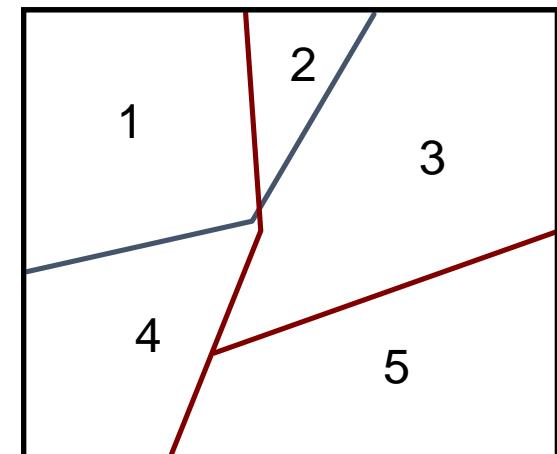
# “overlaps”: Polygon overlay



layer 1



layer 2



overlay

1	A
2	B

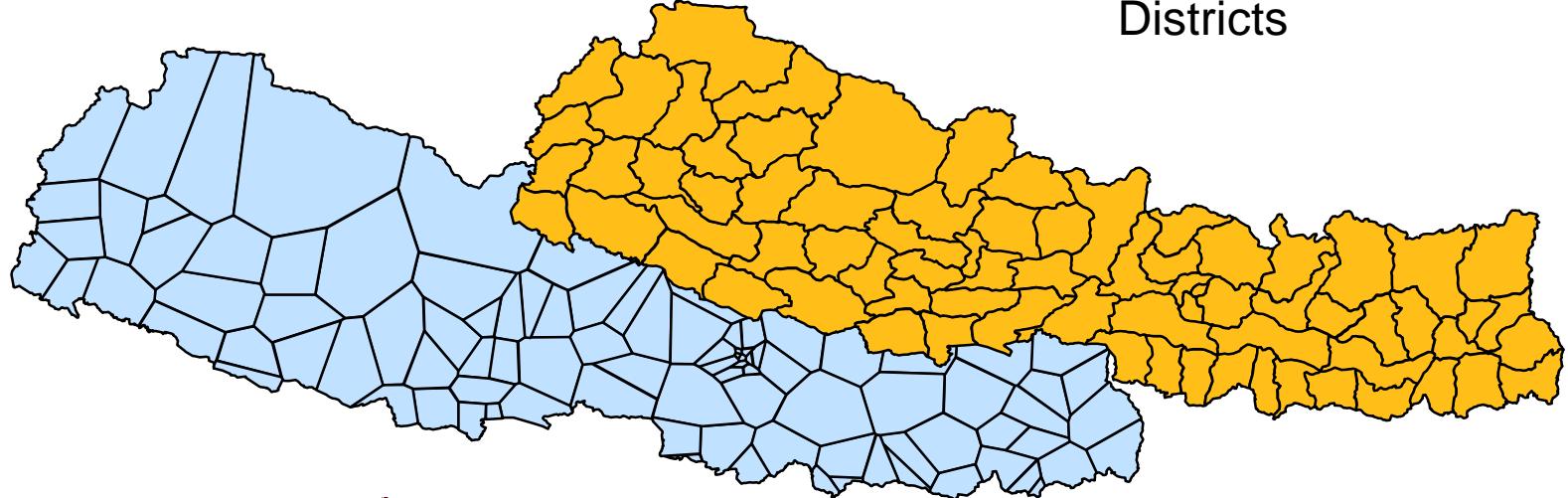
1	I
2	II
3	III

1	A	I
2	A	II
3	B	II
4	B	I
5	B	III

Output layer contains  
all attributes from both input layers

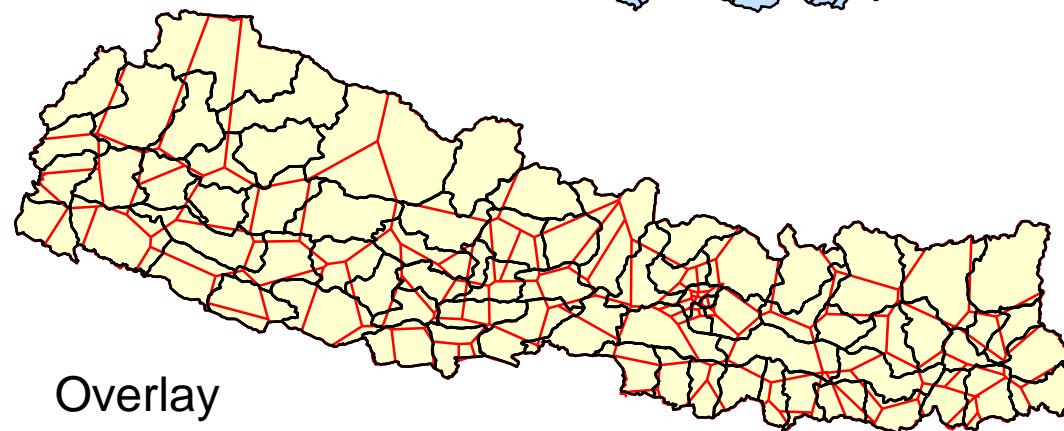
# Polygon overlay

Hospital Catchment  
Areas



Districts

Overlay

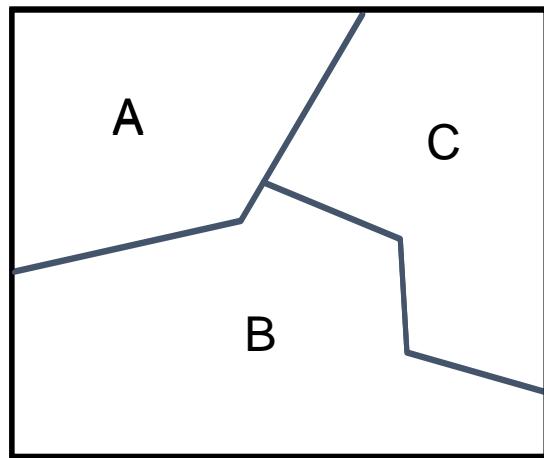


# Areal weighting

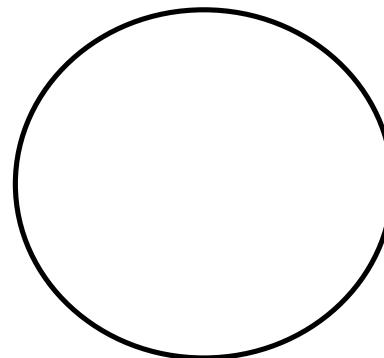
- if 30 % of district  $d$  overlaps with hospital zone  $z$ , then zone  $z$  will also receive 30% of district  $d$ 's population
- areas of overlap derived from a polygon overlay operation
- assumes that districts have constant densities

# Creating subsets

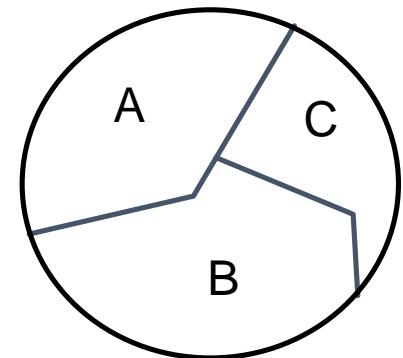
- create a subset of a data set using another incompatible set
- “cookie-cutting”



input



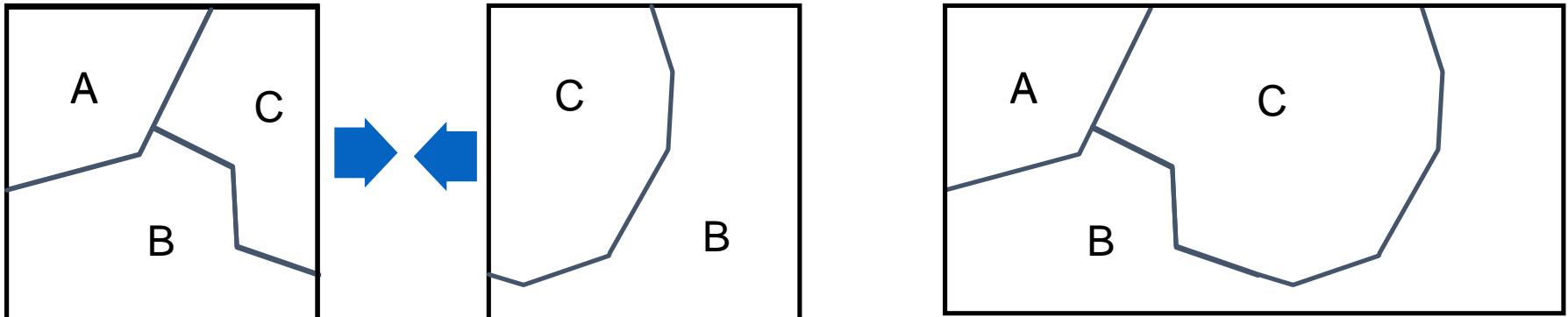
clip cover



output

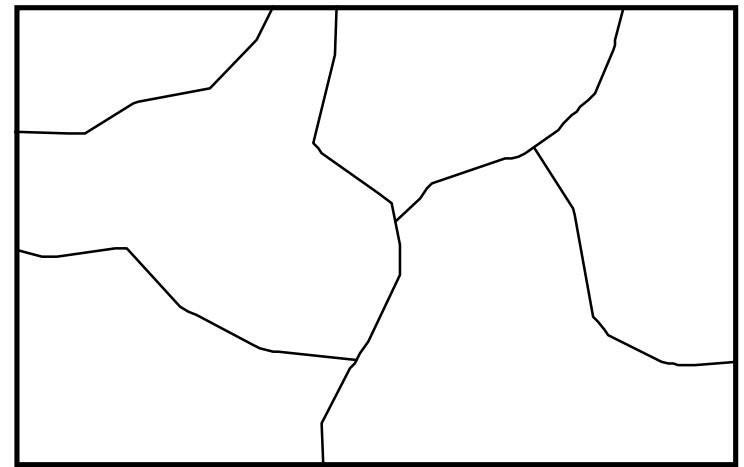
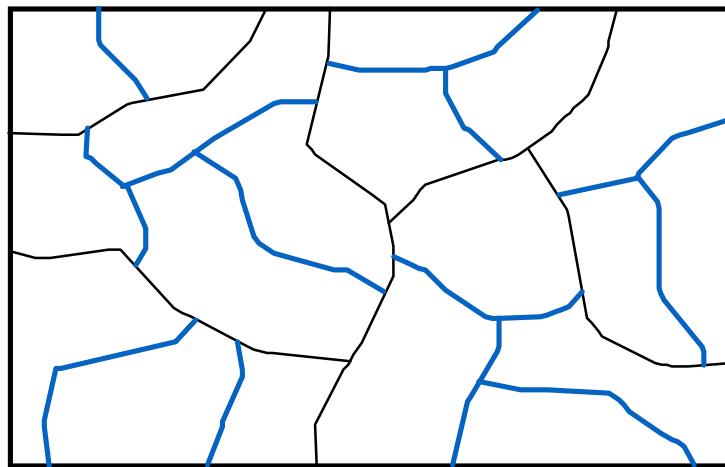
# Appending data sets

- combine data sets that may have been digitized separately



# Merging polygons

- aggregation by deleting internal boundaries



# Merging polygons

- attribute data cannot be merged automatically, since different data types need to be treated differently
  - categorical data: need to use specific rules
  - count data need to be aggregated

# Merging polygons

- some systems only preserve the field that indicates which features to merge (e.g., the state name)
- in others, the user needs to define which method to use to merge data
  - e.g., sum, average, most common value or others

# Chapter 2: Spatial Databases

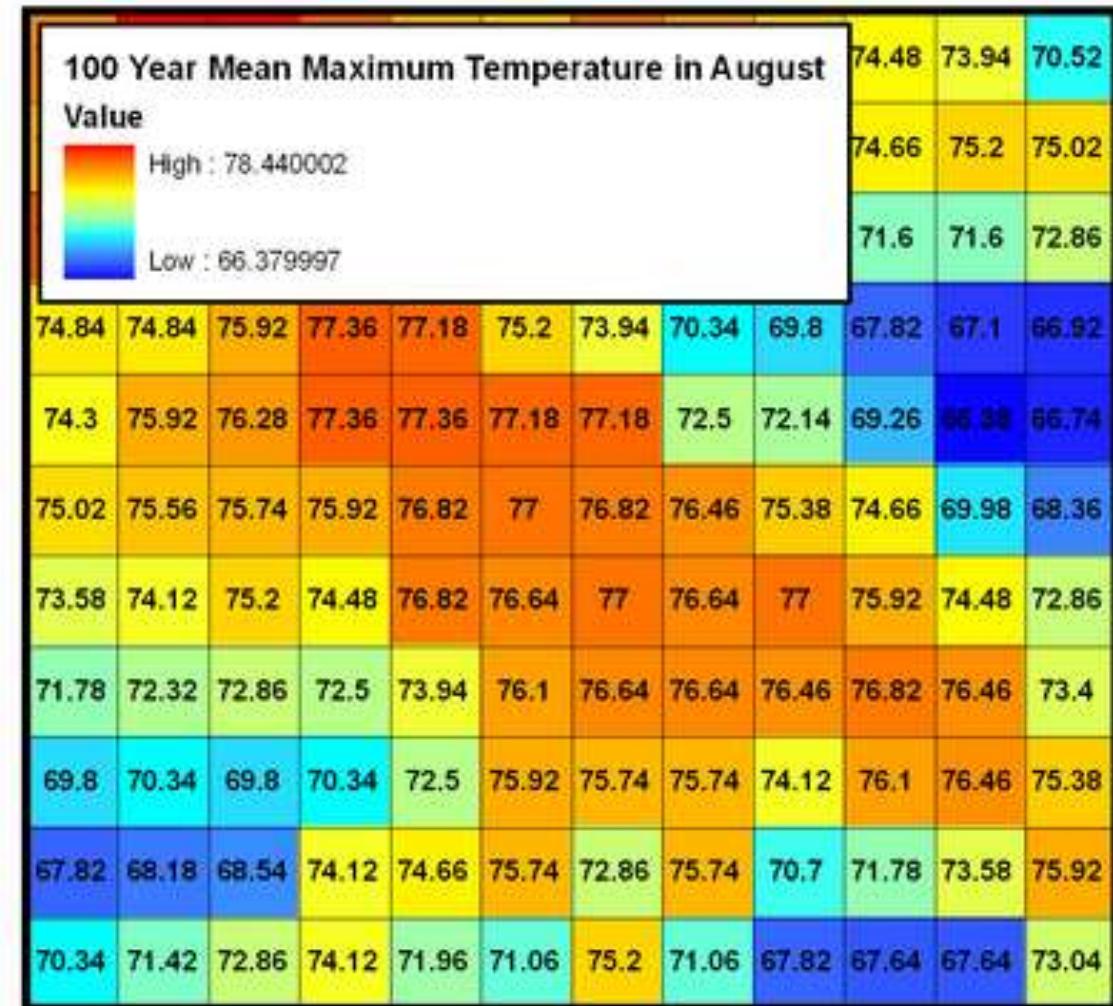
## 2.2. Raster Data Model



QuickBird Satellite Sensor, February 2002  
©2007 - DigitalGlobe All rights reserved

# The Raster Data Model

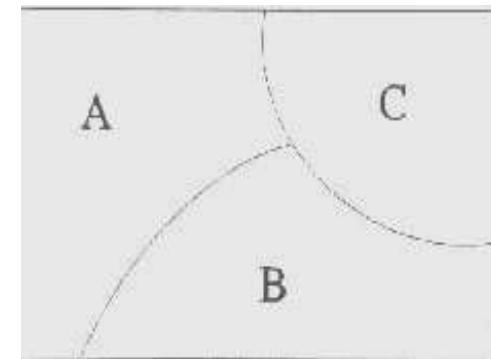
- The Raster Data Model is used to model spatial phenomena that vary continuously over a surface and that do not have discrete dimension
  - Elevation
  - Temperature
  - Rainfall
  - Noise Levels



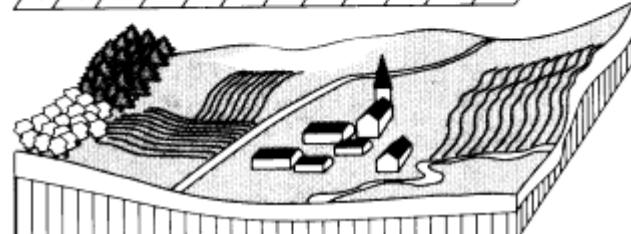
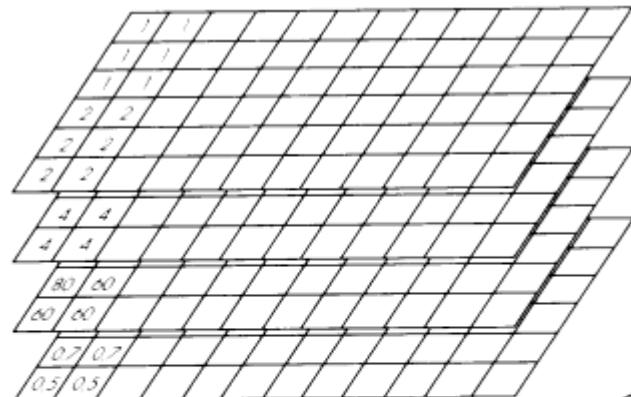
- Raster model divides the entire study area into a regular grid of cells in specific sequence
- the conventional sequence is row by row from the top left corner
- each cell contains a single value .
- one set of cells and associated values is a layer  
there may be many layers in a database, e.g. soil type, elevation, land use, land cover
- A raster representation is composed a series of layers, each with a theme
- Resolution* of a raster is the distance that one side of a grid cell represents on the ground
- The higher the resolution (smaller the grid cell), the higher the precision, but the greater the cost in data storage

1	1	1	2	3
1	1	1	2	3
2	2	2	2	2
4	4	4	2	3
4	4	4	2	3

= grid cell resolution



A	A	A	A	C	C	C	C
A	A	A	A	C	C	C	C
A	A	A	B	B	C	C	C
A	A	B	B	B	B	B	B
A	B	B	B	B	B	B	B



## CELL VALUES: Types of values

- the type of values contained in cells in a raster depend upon both the reality being coded and the GIS different systems allow different classes of values, including :

whole numbers (integers) real (decimal) values

alphabetic values

- many systems only allow integers, others which allow different types restrict each separate raster layer to a single kind of value
- if systems allow several types of values, e.g. some layers numeric, some non-numeric, they should warn the user against doing unreasonable operations

e.g. it is unreasonable to try to multiply the values in a numeric layer with the values in a non- numeric layer

- integer values often act as code numbers, which "point" to names in an associated table or legend
- e.g. the first example might have the following legend identifying the name of each soil class" = 0 :no class" 1 = "fine sandy loam" 2 = "coarse sand" 3 = "gravel "

INTEGER VALUES							
0	0	1	1	1	2	3	
0	1	1	1	2	2	3	e.g. soil class
0	0	1	1	1	2	2	or # of farms
0	0	0	1	1	1	2	
0	0	0	0	0	0	1	

REAL (DECIMAL) VALUES							
2.1	2.3	2.4	2.7	2.9	3.2	3.4	
1.9	2.1	2.4	2.6	3.0	3.3	3.4	e.g elevation
1.8	2.0	2.3	2.5	2.8	3.1	3.2	
1.8	1.9	1.9	2.3	2.5	2.8	3.0	
1.8	1.8	1.8	1.9	2.1	2.3	2.7	

NON-NUMERIC VALUES							
a	a	b	b	c	c	a	
a	b	b	b	c	c	a	e.g. vegetation class
a	b	b	b	b	c	c	
a	a	b	b	b	c	c	
a	a	a	b	c	c	c	

## One value per cell

- Each pixel or cell is assumed to have only one value this is often inaccurate - the boundary of two soil types may run across the middle of a pixel
- in such cases the pixel is given the value of the largest fraction of the cell, or the value of the middle point in the cell
- note, however, a few systems allow a pixel to have multiple values

the NARIS system developed at the University of Illinois in the 1970s allowed each pixel to have any number of values and associated percentages  
e.g. 30% a, 30% b, 40% c

## RUN ENCODING

geographical data tends to be "spatially auto-correlated", meaning that objects which are close to each other tend to have similar attributes

**Tobler** expressed it this way: "All things are related, but nearby things are more related than distant things "

because of this principle, we expect neighboring pixels to have similar values

so instead of repeating pixel values, we can code the raster as pairs of numbers - (run length, value)

e.g. instead of 16 pixel values in original raster matrix, we have :

4A 1A 3B 2A 2B 3A 1B produces 7 integer/value pairs to be stored

if a run is not required to break at the end of each line we can compress this further :  
5A 3B 2A 2B 3A 1B = 6 pairs

A	A	A	A
A	B	B	B
A	A	B	B
A	A	A	B

## Row order

are there better ways of ordering the raster than row by row from the top left?

other orders may produce greater compression

## Row prime order

suppose we reverse every other row :diagram

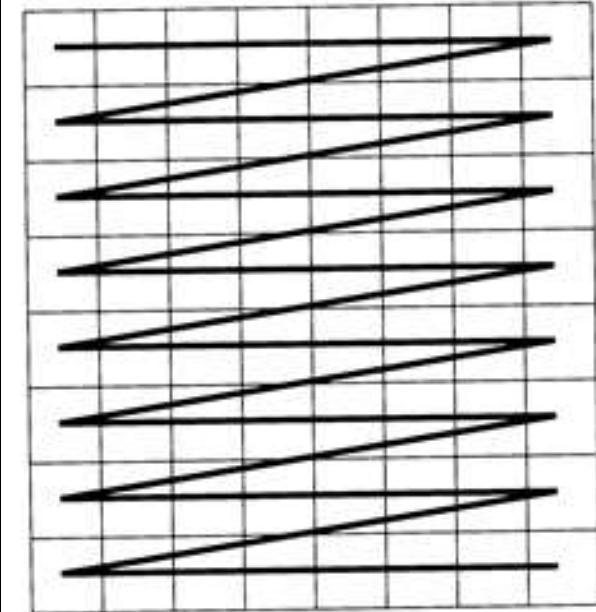
this has the charming name boustrophedon from the Greek for "how an oxen plows a field "

avoids a long jump at the end of each row, so perhaps the raster would produce fewer runs and thus greater compression

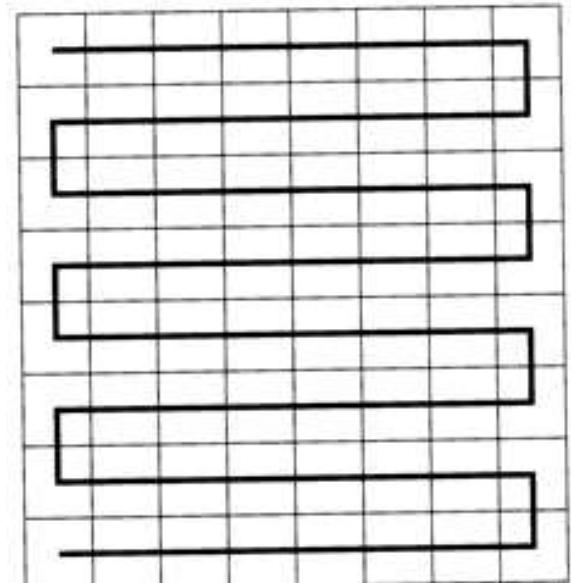
this order is used in the Public Land Survey System: the sections in each township are numbered in this way

## Morton order

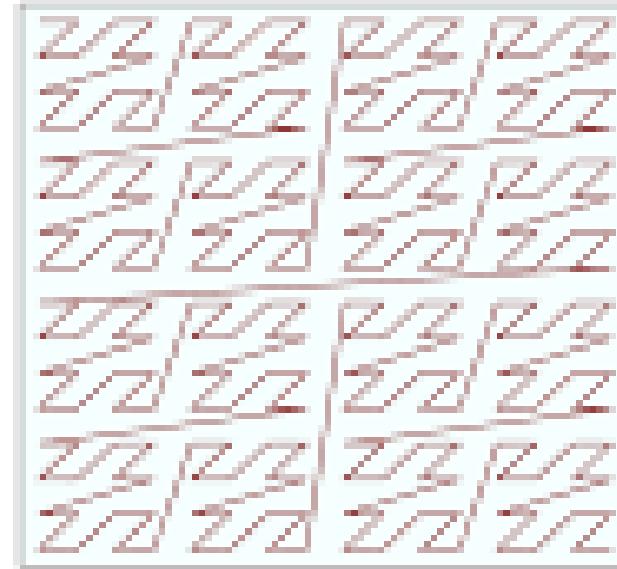
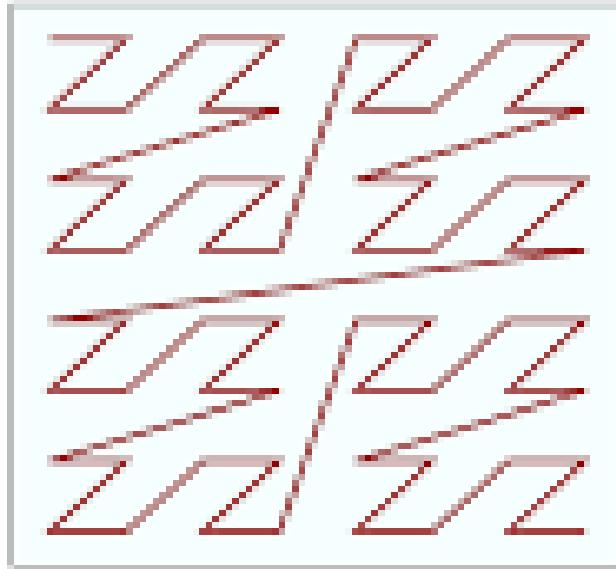
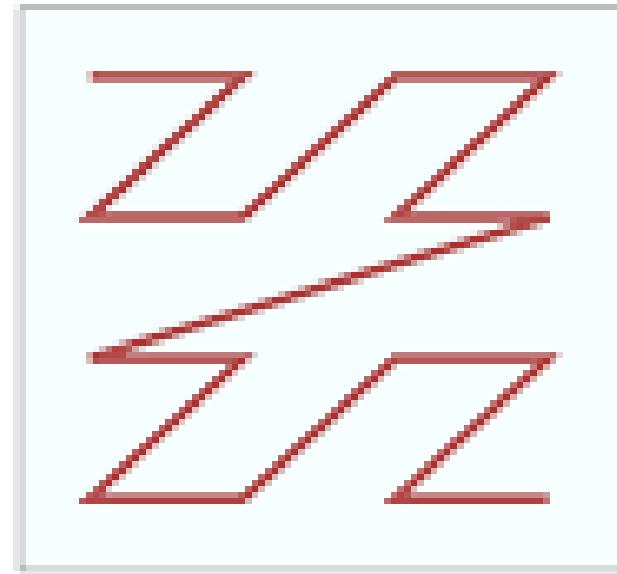
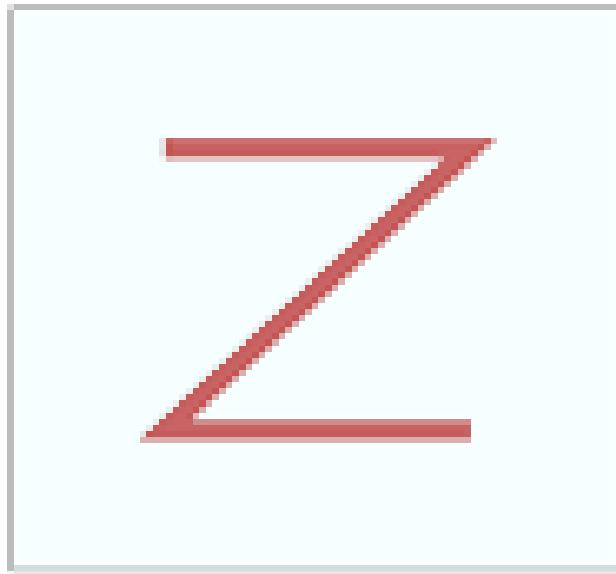
Morton order is the basis of many efforts to reduce database volume named for Guy Morton who devised it as a way of ordering data in the Canada Geographic Information System however, this way of ordering or scanning a raster was well known long before Morton it is associated with the names of several mathematicians and geometers: Hilbert,



a) row

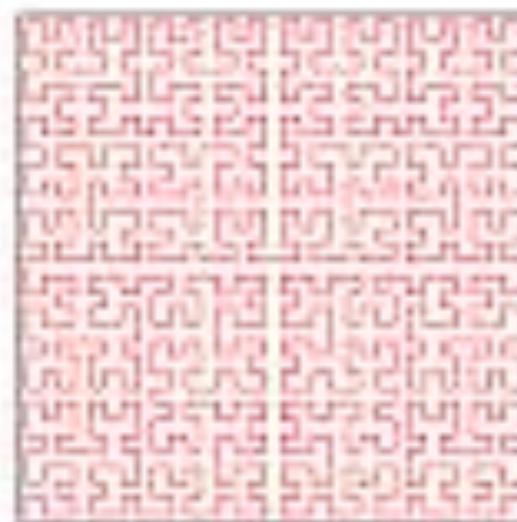
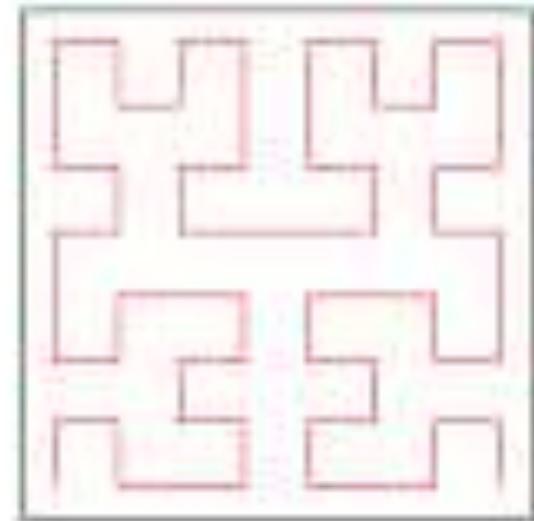


b) row-prime



## Peano scan (also Pi-Order or Hilbert)

the Peano scan or Pi-order is like boustrophedon in always moving to a neighboring pixel diagram



# DECODING SCAN ORDERS

**Since Morton and Peano orders are useful but complex, two types of questions arise when they are used :**

- ❑ What are the row and column numbers for a given pixel ?
  - ❑ What is the position in the scan order for a given row and column number?

## 1- How to go from row 2, column 3 to Morton sequence?

**a. convert row and column numbers to binary representations :**

b. interleave the bits, alternating row and column bits (called bit interleaving) =  
**1101**

c. evaluate this sequence of bits as a binary number      Answer:  $8 + 4 + 1 = 13$

## 2- How to find row and column number from Morton position 9?

a. convert the position number to a binary number ( $8 + 1 = 9$ ) = 1 0 0 1

**b. separate the bits :**

row 10 ≡ 2      col 01 ≡ 1

## Generalization

- We can express the row and column number to any base, not just base 2 (binary), and including mixtures of bases

**example:** row 6, column 15,  
using base 4 instead of base 2

**64s    16s    4s    1s**

$$\text{row } 6 = 1 \times 4 + 2 \times 1 = 12$$

$$\text{col } 15 = 3 \times 4 + 3 \times 1 = 33$$

interleaving:      1323

$$1 \times 64 + 3 \times 16 + 2 \times 4 + 3 \times 1 = 123$$

**answer: row 6 column 15 is position 123**

# HIERARCHICAL DATA STRUCTURES

## A. INTRODUCTION

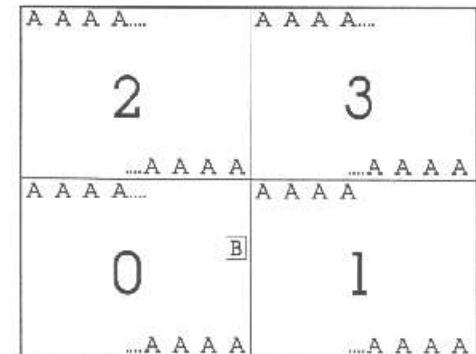
different scan orders produce only small differences in compression

the major reason for interest in Morton and other hierarchical scan orders is for faster data access

the amount of information shown on a map varies enormously from area to area, depending on the local variability

it would make sense then to use rasters of different sizes depending on the density of information

large cells in smooth or unvarying areas, small cells in rugged or rapidly varying areas



unfortunately unequal-sized squares won't fit together ("tile the plane") except under unusual circumstances one such circumstance is when small squares nest within large ones

there are, however, some methods for compressing raster data that do allow for varying information densities

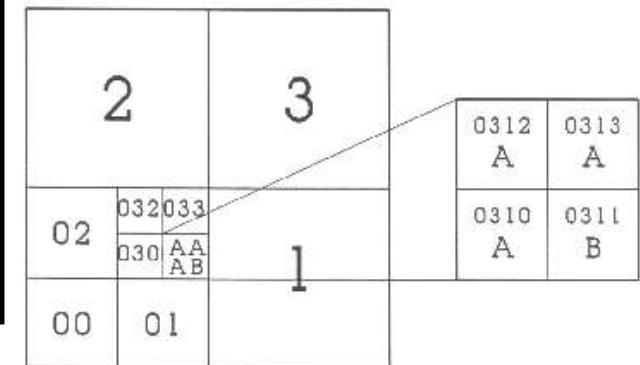
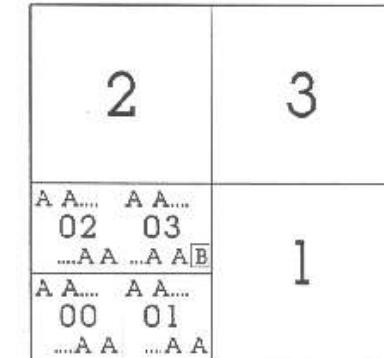
## B. INDEXING PIXELS

consider the 16 by 16 array in which just one cell is different notation: row and column numbering starts at 0 thus the odd cell is at row 4, column 7

## Procedure

begin by dividing the array into four 8x8 quadrants, and numbering them 0, 1, 2 and 3 as in the Morton order quads 1, 2 and 3 are homogeneous (all A) quad 0 is not homogeneous, so we divide only it into four 4x4 quads these are numbered 00, 01, 02 and 03 because they are partitions of the 8x8 quad 0 of these, 00, 01 and 02 are homogeneous, but 03 is divided again into 030, 031, 032 and 033 now only 031 is not homogeneous, so it is divided again into 0310, 0311, 0312 and 0313

what we have done is to recursively subdivide using a rule of 4 until either: a square is homogeneous or



we reach the highest level of resolution (the pixel size)  
this allows for discretely adaptable resolution where each resolution step is fixed. this concept is related to the use of Morton order for run encoding if we had coded the raster using Morton order, each homogeneous square would have been a run 8x8 squares are runs of 64 in Morton order, 4x4 are runs of 16, etc the run encoded Morton order would have been: 16A 16A 16A 4A 1A 1B 1A 1A 4A 4A 64A 64A 64A if we allow runs to continue between blocks we could reduce this to: 53A 1B 202A.

## Decoding locations

the conversion to row and column is the same as for decoding Morton numbers except that in this case the code is in base 4 in the example the lone B pixel is assigned code 0311

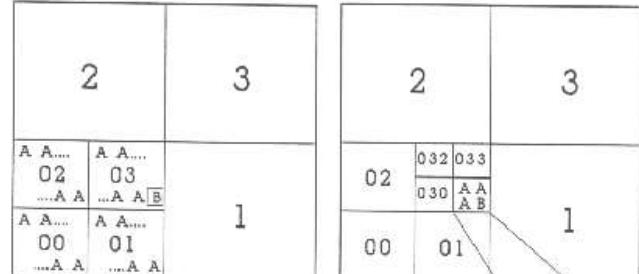
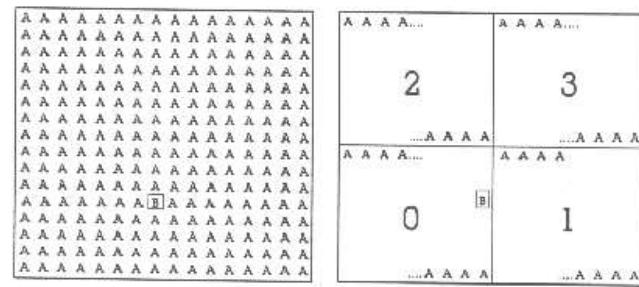
1. convert the code to base 2

hint: every base 4 digit converts to a pair of base 2 digits      thus 0311 becomes 00110101

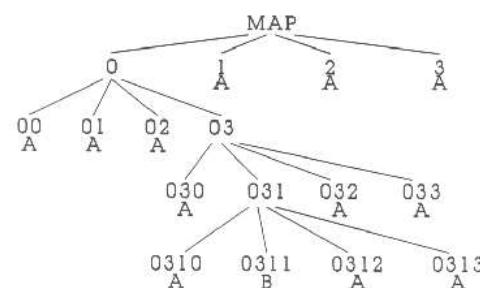
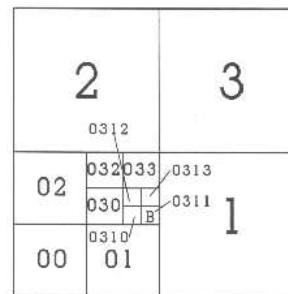
2. separate the bits to get:

row 0100 = 4

column 0111 = 7



0312	0313
A	A
0310	0311
A	B



so the numbering system is just the Morton numbering of blocks, expressed in base 4

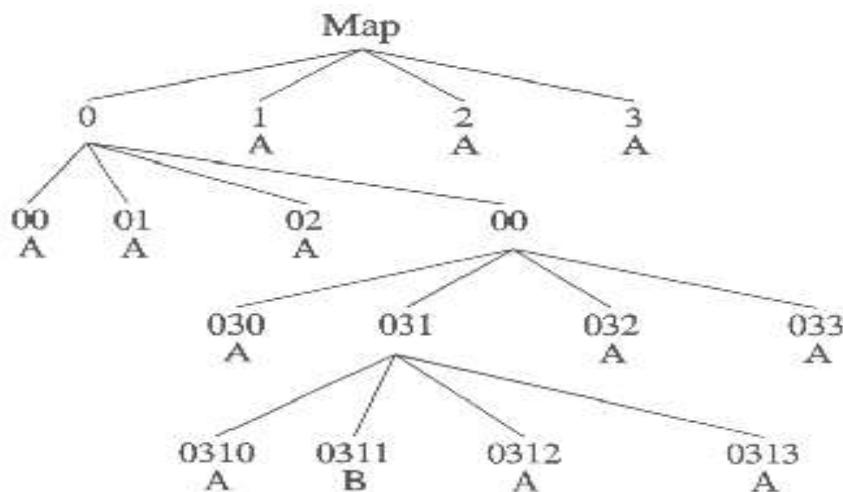
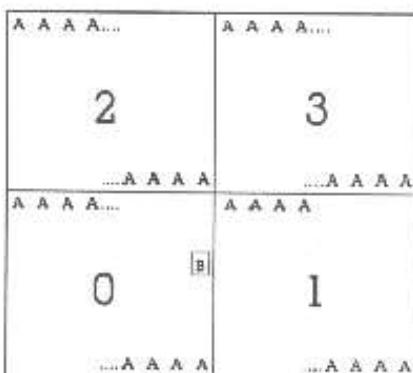
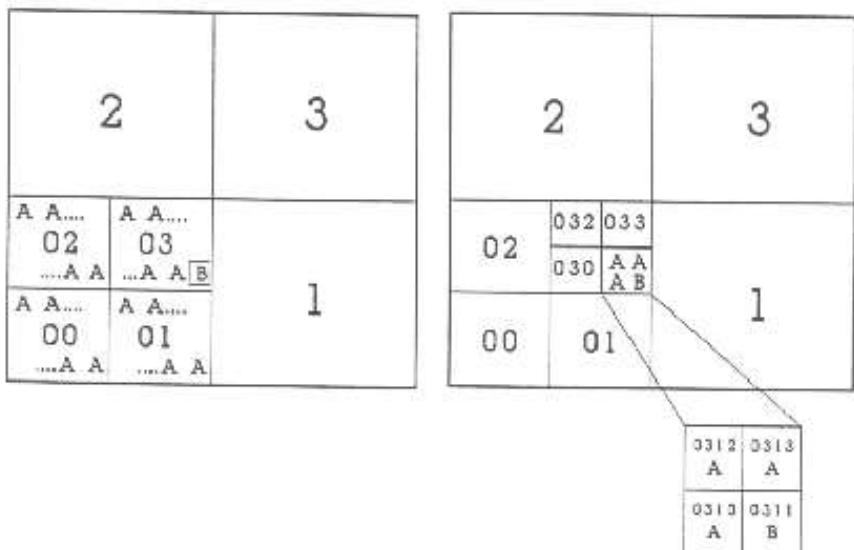
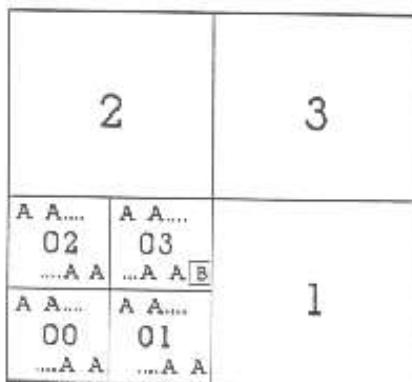
however, sequence and data compression are not the most useful aspects of this concept

## C. THE QUADTREE

can express this sequencing as a tree the top is the entire array at each level there is a four-way branching each branch terminates at a homogeneous block

the term quadtree is used because it is based on a rule of 4 each of the terminal branches in the tree (the ones having values) is known as a leaf in this case there are 13 leaves or homogeneous square blocks

A 10x10 grid of the letter 'A' in a standard font. The letters are arranged in a continuous pattern across the grid. The 10th letter in the 10th row is highlighted with a solid black square.



LEVEL

0	pointer			
1	pointer	A	A	A
2	A	A	A	pointer
3	A	pointer	A	A
4	A	B	A	A

Position:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Contents:	2	6	A	A	A	A	A	A	10	A	14	A	A	B	A	A	
(level):	0	1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	

## Coding quadtrees

to store this tree in memory, need to decide what to store in each memory location

there are many ways of storing quadtrees, but they all share the same basic ideas

one way is to store in each memory location EITHER: 1. the value of the block (e.g. A or B), or 2. a pointer to the first of the four "daughter" blocks at the next level down

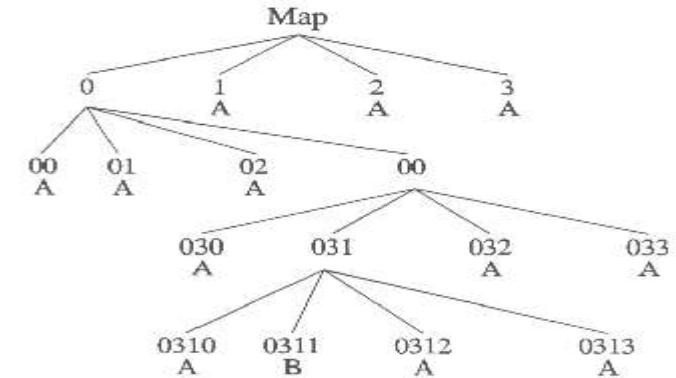
all four daughter blocks of any parent always occur together overhead - Coding quadtrees

thus, the quadtree might be stored in memory as:

Position: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

Contents: 2 6 AAAAAAA 10 A 14 AAA B AA

(level): 0 1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4



LEVEL

0	pointer	A	A	A
1	pointer	A	A	A
2	A	A	A	pointer
3	A	pointer	A	A
4	A	B	A	A

Position: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

Contents: 2 6 AAAAAAA 10 A 14 AAA B AA

(level): 0 1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4

the content of position 1 is a pointer indicating that the map is subdivided into four blocks whose contents can be found starting at position 2 position 2 indicates that the four parts of the 0 block can be found beginning at position 6 positions 3, 4 and 5 indicate that the other three level 1 blocks are all A and are not further subdivided

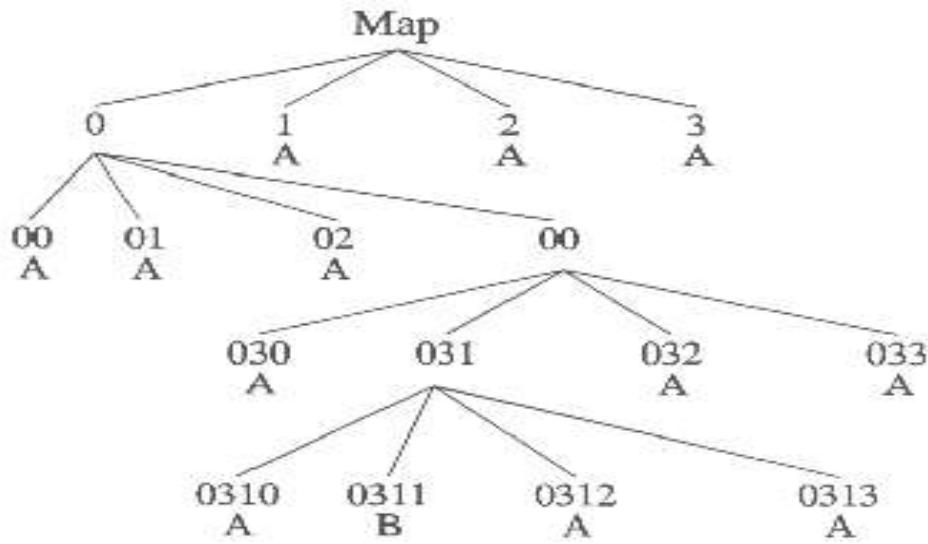
## Accessing data through a quadtree

consider two ways in which this quadtree may be accessed:

1. find all parts of the map with a given value
2. determine the contents of a given pixel

notation: if the array has  $2n$  by  $2n$  pixels there are  $n$  possible levels in the tree, or  $n+1$  if we count the top level (level 0) use  $m$  for the number of leafs

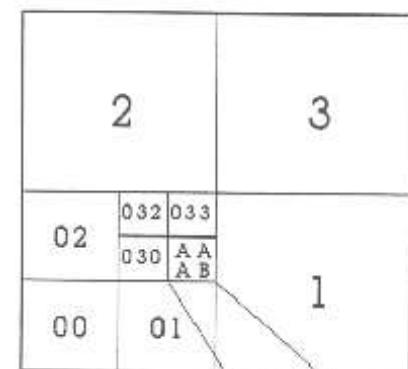
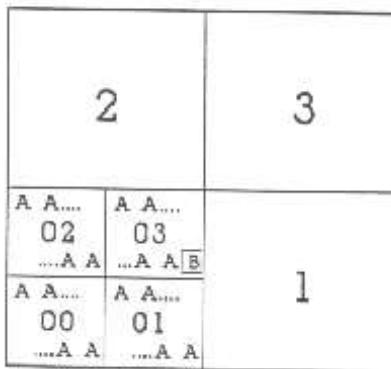
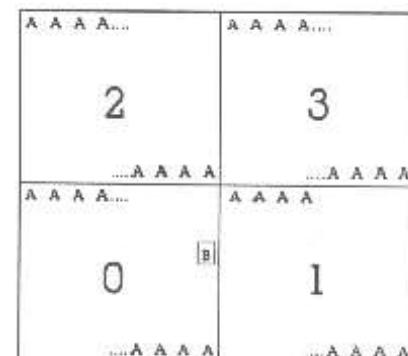
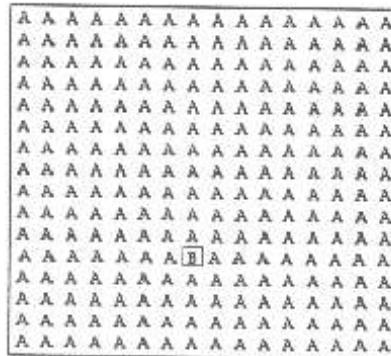
Map



## LEVEL

0	pointer				
1	pointer	A	A	A	pointer
2	A	A	A	A	
3	A	pointer	A	A	
4	A	B	A	A	

Position:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Contents:	2	6	A	A	A	A	A	10	A	14	A	A	B	A	A		
(level):	0	1	1	1	1	2	2	2	3	3	3	3	4	4	4	4	4



0312	0313
A	A

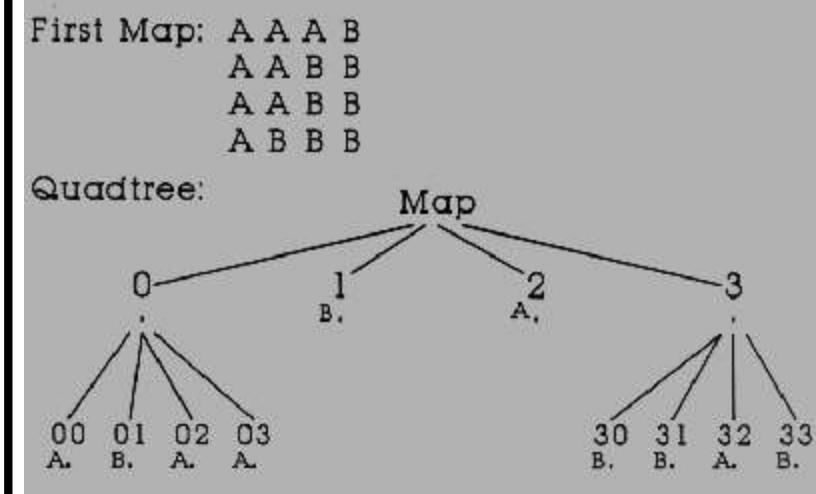
0313	0311
A	B

1. to find the parts of a map with a given value we must examine every leaf to see if its value matches the one required this requires  $m$  steps as there are  $m$  leafs
2. to find the contents of a given pixel, start at the top of the tree if the entire map is homogeneous, stop as the contents of the pixel are known already if not, follow the branch containing the pixel do know which branch to follow: take the row and column numbers, write them in binary, interleave the bits, and convert to base 4 e.g. row 4, column 7 converts to 0311 at each level, use the appropriate digit to determine which branch to follow e.g. for 0311, at level 0 follow branch 0, at level 1 follow branch 3, etc. in the worst case, may have to go to level  $n$  to find the contents of the pixel, so the number of steps will be  $n$

# QUADTREE ALGORITHMS AND SPATIAL INDEXES

This unit examines how quadtrees are used in several simple processes, including:

- Measurement of area
- Overlay
- Finding adjacent leafs
- Measuring the area of contiguous patches



## Measurement of area ALGORITHM:

to measure the area of A on the map:

traverse the tree and add those leafs coded A, weighted by the area at the level of the leaf

### Example

in the example quadtree, elements at level 0 have area 16, at level 1 - area 4, at level 2 - area 1  
thus, area of A is: 1 (leaf 00) + 1(leaf 02) + 1 (leaf 03) + 4

(leaf 2) + 1 (leaf 32) = 8 units

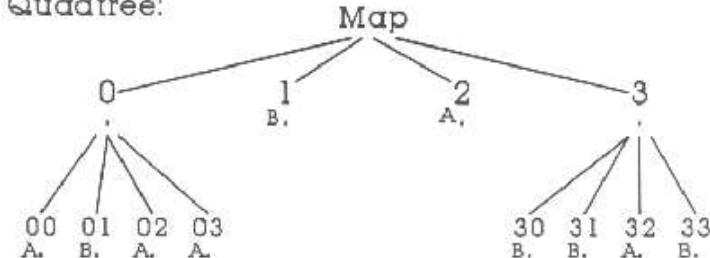
# OVERLAY ALGORITHM

To overlay the two maps:

- traverse the trees simultaneously, following all branches which exist in either tree where one tree lacks branches (has a leaf where the other tree has branches)
  
- assign the value of the associated leaf to each of the branches  
e.g. node 3 is branched on map 1, not on map 2 the leafs derived from this node (30, 31, 32 and 33) have values B, B, A and B on map 1, all 2 on map 2 the new tree has the attributes of both of the maps, e.g. A1, B2

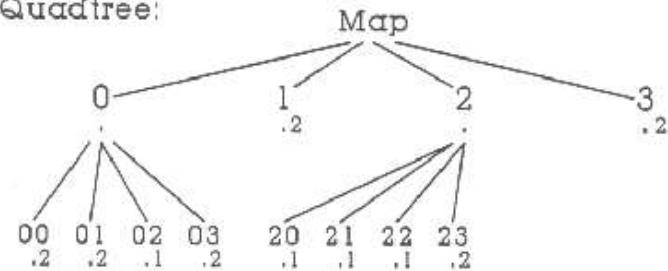
First Map: AAA B  
AAB B  
AAB B  
ABB B

Quadtrees:



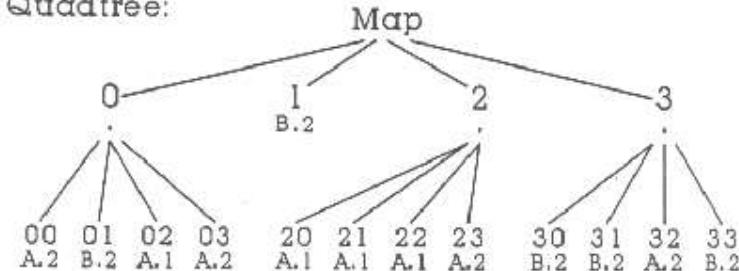
Second Map: 1 2 2 2  
1 1 2 2  
1 2 2 2  
2 2 2 2

Quadtrees:



First Map: AAA B      Second Map: 1 2 2 2  
AAB B      1 1 2 2  
AAB B      1 2 2 2  
ABB B      2 2 2 2

Quadtrees:



## D. ADJACENCY ALGORITHM

Find if two leafs (e.g. 03 and 2) are adjacent Or, find the leafs adjacent to a given leaf (e.g. 03)

note that in arc based systems adjacencies are coded in the data structure (R and L polygons), so this operation is simpler with vector based systems

### Definition

Here adjacent means sharing a common edge, not just a common point diagram

### Two cases

leaf codes are:

1. same length (same size blocks, e.g. 01 and 02) or
2. one is longer than the other (different size blocks, e.g. 03 and 2)

### **Solving this problem requires the use of:**

1. conversion from base 4 to binary and back base 4 because of the "rule of 4" used in constructing quadtrees
2. bit interleaving
3. a new concept called Tesserel Arithmetic

## Tesseral Arithmetic

- Tesseral arithmetic is an alternate arithmetic useful for working with the peculiarities of quadtree addressing
- To add binary numbers normally, a "carry" works to the position to the left
  - e.g. adding 1 to 0001 gives 0010
  - this is the same as decimal arithmetic except that carries occur when the total reaches 2 instead of 10
- In tesseral arithmetic, a "carry" works two positions to the left
  - e.g. adding 1 to 0001 gives 0100
- The reverse happens on subtraction
  - 1000 less 1 is 0010 not 0111, as the subtraction affects only the alternate bits
- In other words, if we number the bits from the left starting at 1
  - adding or subtracting 1 affects only the even-numbered bits
  - adding or subtracting 2 (binary 10) affects only the odd-numbered bits

## Determining Adjacency

1. same size blocks: two leafs are adjacent if their binary representations differ by binary 1 or 10 (decimal 1 or 2) in tesseral arithmetic

example: 01 and 03 are adjacent because 0001 and 0011 differ by binary 10, or decimal 2

example: 033 and 211 are adjacent because in tesseral arithmetic  $001111 + 10 = 100101$ , or  $100101 - 10 = 001111$

2. different size blocks: taking the longer of the two codes:  
convert it from base 4 to binary

tesseral-add and -subtract 01 and 10 to create four new codes  
reject any cases where subtracting was not possible (a "negative"  
code would have resulted, or a "carry" would have been necessary to  
the left of the leftmost digit)

discard the excess rightmost digits in the resulting transformed longer  
codes

convert back to base 4 to get the leaf

the two blocks are adjacent if any of the transformed and truncated codes are equal to the shorter code

**example:** Are 02 and 2 adjacent?

convert 02 to binary = 0010

$0010 + 1 = 0011$

$0010 + 10 = 1000$

$0010 - 1$  (impossible)

$0010 - 10 = 0000$

truncating gives 00 and 10

these are equal to 0 and 2 in base 4

therefore, 02 and 2 are adjacent (also 02 and 0 are adjacent)

**example: Are 033 and 2 adjacent?**

convert 033 to binary = 001111

$001111 + 1 = 011010$

$001111 + 10 = 100101$

$001111 - 1 = 001110$

$001111 - 10 = 001101$

truncating to two digits gives 01, 10 and 00

these are equal to 1,2 and 0 in base 4

therefore, 033 and 2 are adjacent

**example: Find leafs adjacent to 03 in the first map above**

method: find the codes of adjacent blocks of the same size, then work down the tree to find the appropriate leaf

(note: can only find equal or shorter codes - equal or bigger leaf blocks)  $0011 + 1 = 0110 = 12$  : leaf 1  $0011 + 10 = 1001 = 21$  : leaf 2  $0011 - 1 = 0010 = 02$  : leaf 02  $0011 - 10 = 0001 = 01$  : leaf 01

## E. AREA OF A CONTIGUOUS PATCH ALGORITHM

- Find the area of a contiguous patch of the same value, e.g. all A Corollary:  
How many separate patches of A are there?
- Note: this is a general method which can be used in both quadtree and vector data structures
  - i.e. find contiguous sets of quadtree blocks or irregularly shaped polygons, given that adjacencies are known or can be determined
  - the following example uses the original raster map
  - note that there are only two contiguous patches; the areas of A and B form only one patch each

## Area of a contiguous patch

- Create a list of leafs, with their associated codes, by traversing the tree allow space for a "pointer" for each leaf, and give it an initial value of 0

**for each leaf i:**

- find all adjacent leafs j with equal or shorter length codes (4 maximum)
- if the adjacent leaf j has the same value, determine which of i and j has the higher (larger value) position in the list, and set its pointer to the lower position
- if a pointer has already been changed, it may be changed again or left, the result is the same)
- This produces the final pointer list

## Results

1. the number of contiguous patches will be equal to the number of zeros in the example, two pointers are zero, indicating two contiguous patches
2. the value of each patch can be obtained by looking up the values of leafs with 0 pointers  
in the example, leafs 00 and 01 have 0 pointers  
these have the values A and B respectively
3. to find the area of each patch, select one of the zeros and sum its area plus the areas of any leafs which point to it directly or indirectly the component leafs of each patch can be found by starting at with a leaf at the end (or beginning) of the list and following the pointers until a 0 is found

# NUMBER OF A CONTIGUOUS PATCH ALGORITHM

Algorithm Steps:

1. Construct the pointer table as follows:

2. For each leaf in the tree, do the following:

- a. Get all adjacent leaves.
- b. Neglect different values.

c. For each adjacent leaf which has the same value, do the following:

I. If the adjacent leaf address is less than the current leaf address [like (022) is less than (200)], then neglect this adjacent leaf.

II. If the adjacent leaf address is greater than the current leaf address[ like (202) is greater than (200)], then put the ID of the current leaf in the pointer of this adjacent leaf.

III. If there are more than one adjacent leaves which their addresses are greater than the current leaf address[ like (201) and (202) are greater than (200)], then put the ID of the current leaf in the pointer of the adjacent leaf that has lower address[means for the example above: put the ID of (200) in the pointer of the (201)].

IV. If there are no other adjacent leaves that has the same value for the current leaf, then do nothing.

ID	Address	Level	Value	Pointer
1	000	3	A	0
2	001	3	B	0
3	002	3	C	0
4	003	3	B	0
5	01	2	C	0
6	...	...	...	0

# AREA OF A CONTIGUOUS PATCH ALGORITHM

**For each different path,** do the following:

For each different value:

$$\text{Area} = \sum_L \text{ number of leaves in certain level } [L] * \text{level size.}$$

Where:

$$\text{Level size} = 2^{2(K-L)}$$

K is base of the map like(Map[4X4] its base is 2 and Map[8X8] its base is 3)

L is the current level.

# K-d trees

## 1. Node structure

- In the **2-d tree** each node has a certain structure these records has the following type

**Nodetype**=record

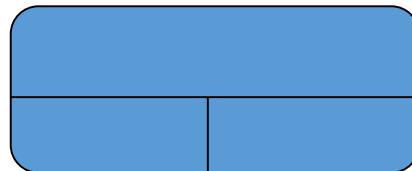
**INFO**:infotype

**XVAL**: real

**YVAL**:real

**LLink** :node type

**RLINK**: node type



We draw a vertical **or** horizontal lines at the node depends on the level of the node is even or odd

# Point Quadtrees

## 1. Node structure

- In the **Quadtrees** tree each node has a certain structure these records has the following type:

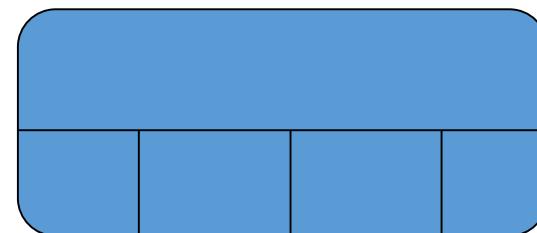
**QNodetype**=record

**INFO**:infotype

**XVAL**: real

**YVAL**:real

**NW;SW;NE;SE**: qnode type



At any node we divide the image into four parts by drawing a vertical **and** horizontal lines at the node

# K-d trees

## 1. Node structure (cont.)

Suppose T is q pointer to the root of 2-d tree. If N is a node in this tree then the level of node N is defined as :

$$\text{Level (N)} = \begin{cases} 0 & \text{if } N \text{ is the root of the tree} \\ \text{Level (P)} + 1 & \text{if } N \text{'s parent is P} \end{cases}$$

- If 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 sub tree rooted at N.RLINK has the property that ***P.XVAL > N.XVAL***
- If 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 sub tree rooted at N.RLINK has the property that ***P.YVAL > N.YVAL***

# Point Quadtrees

- The point quadtree like the 2-d tree is used to represent point data in two dimensional spaces
- unlike the 2-d tree point quadtrees always split regions into four parts

## K-d trees

### 3. Deletion in 2-d Trees

- the most complex part of dealing with a 2-d tree is deletion of point from the tree.
- Suppose T is a 2-d tree and (x,y) refers to a point that we wish to delete from the tree.
- the first step in deletion is to search for the node N in T that has N.XVAL=x and N.YVAL=y
- if N is a **leaf node** then the deletion of N is easy we merely set the appropriate field (LLINK,RLINK) of N's parent to nil and return N to available storage
- If N is an **interior node** in this case either the subtree rooted at N.LLINK which we will denote by Tl or the subtree rooted at N.RLINK will denote by Tr is nonempty what we like to do is to find a node R from either Tl or Tr that can replace node N and that can subsequently be deleted fro; the subtree

## Point Quadtrees

### 3. Deletion in point Quadtree

- In order to successfully apply the deletion technique to a point quadtree
- when deleting an interior node N
- We must find a replacement node R in one of the subtree of N

## K-d trees

### 3. Deletion in 2-d Trees

- **Step 1:** find a candidate replacement node R that occurs in  $T_i$  for  $i \in (l, r)$
- **Step 2 :** Replace all of N's nonlink fields by those of R
- **Step 3:** Delete R from  $T_i$

### 4. Range queries in 2-D trees

- the range query 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  $r$  of  $(x_c, y_c)$
- When processing a range query it is helpful to recall that each node N in 2-d tree implicitly represents a region  $R_n$
- This is useful because 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

## Point Quadtrees

When inserting a node into the tree, we need to ensure the following

1. if N is the root of the tree T, then  $N.XLB = -\infty$  and  $N.YLB = -\infty$ ;  $N.XUB = +\infty$ ;  $N.YUB = +\infty$
2. If P is the parent of  $N_m$  then the following table

## K-d trees

### 4. Rang queries in 2-D trees (cont.)

- In general each node N has at most four associated constraints that jointly define the region represented by the node
1. XLB this constraint represents the lower bound on x and has the form  $x \geq c_1$
  2. XUB this constraint represents the Upper bound on x and has the form  $x < c_2$
  3. YLB this constraint represents the lower bound on y and has the form  $y \geq c_3$
  4. YUB this constraint represents the lower bound on y and has the form  $y < c_4$

## Point Quadtreees

### 4. Rang queries in Point Quadtreees

- Rang queries in point Quadtreees are treated in almost exactly the same way as they are treated in 2-d trees
- Each node in a point Quadtreees represents a region
- the approach toward computing a rang query avoids searching subtree rooted at nodes whose associated region has no intersection with the circle defined by the rang query

## K-d trees

### 4. Range queries in 2-D tree

1. the root of the tree has XLB and YLB set to  $-\infty$  and XUB and YUB set to  $+\infty$
2. If node N has node P as its parent and level (p) is even then

N.XLB = P.XLB if N = P.LLINK

N.XLB= P.XVAL if N = P.RLINK

N.XUB= P.XVAL if N = P.LLINK

N.XYB = P.XUB if N=P.RLINK

N.YLB=P.YLB

N.YUB=P.YUB

3. If node N has node P as its parent and level (p) is odd then X will change with Y

## The MX-Quadtree

- In the case of both k-d trees and point Quadtree the shape of the tree depends upon the order in which objects are inserted into the tree
- In particular the order affects the height of the tree which in turn may affect the complexity of search and insertion operation
  - Also for both k-d trees and point Quadtree each node N represents a region and splits the region into two in case of 2-d tree and four in case of point Quadtree
  - the split may be depending upon exactly where the point (N.XVAL and N.YVAL) is located inside the region represented by node N

# The MX-Quadtree

1. In Contrast the aim behind The MX-Quadtree was to ensure that the (shape and height) of the tree was independent of the number of nodes present in the tree
2. As well as the order of insertion of these node additionally the The MX-Quadtree aimed at providing efficient deletion and search algorithms
- 3. The MX-Quadtree work as follows

- First we assume that the map being represented is split up into a grid of size  $(2^k \times 2^k)$
- The application developer is free to choose k to reflect the desired granularity  
But once k is chosen it must be kept fixed
- MX-Quadtree have exactly the same node structure as Point-Quadtree that is they have the type newqtnodetype

There is one difference thought the root of an MX-Quadtree represents the region specified by  $XLB=0$  and  $XUB= 2^k$   $YLB=0$  and  $YUB= 2^k$

Furthermore when a region is split it is split down the middle

If N is a node then the region represents by four children as the following table

## The MX-Quadtree

### Insertion/search in The MX-Quadtree

- Each point  $(x,y)$  in an MX-Quadtree represents the  $1 \times 1$  region whose lower-left corner is  $(x,y)$
- A point is inserted at the node representing the  $1 \times 1$  region corresponding to that point
- Suppose now that we wish to insert the point A,B,C and D we proceed the follows

1- The insertion of point A with coordinates  $(1,3)$  causes the following

- the root node represents the entire region and A lies in its NW quadrant
- Thus the root's NW child corresponds to the  $2 \times 2$  region whose lower-left corner is the point  $(0,2)$
- The point A is NE subquadrant of this region
- Note that point A is inserted at level 2 in the tree and this level is identical to k in general point will always be inserted at level k in MX-Quadtree

2- The insertion of point B with coordinates  $(3,3)$  causes a branch to the NE quadrant as B Thus the root's NE child corresponds to the  $2 \times 2$  region whose lower-left corner is the point  $(2,2)$  the point B is in the NE subquadrant of the region

## The MX-Quadtree

### Insertion/search in The MX-Quadtree

3. The insertion of the point C with coordinates (3,1) proceeds as follows

- c is in SE quadrant of the whole region
- This causes us to create a new node that is the SE child of the root
- C is in the NE subquadrant of this node

### Deletion IN MX-Quadtree

- Deletion in MX-Quadtree is a fairly simple operation because all point are represented in the leaf level

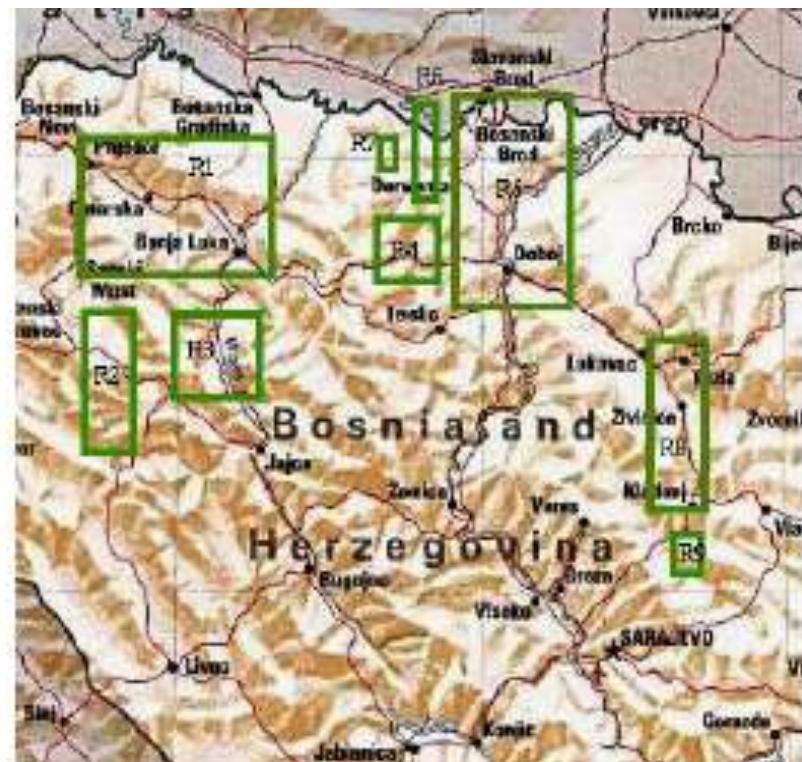
### Rang queries MX-Quadtree

Rang queries in MX-Quadtree are treated in almost exactly the same way as they are treated in point Quadtrees

# 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

**nodetype = record**

**INFO: infotype;**

**XVAL: real;**

**YVAL: real;**

**LLINK: ↑ nodetype**

**RLINK: ↑ nodetype**

**end**

<b>INFO</b>	<b>XVAL</b>	<b>YVAL</b>
<b>LLINK</b>		<b>RLINK</b>

# Node Structure

- *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  $N.\text{LLINK}$  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-d trees, formally

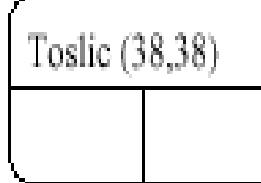
2. If  $N$  is a node in the tree such that  $\text{level}(N)$  is odd, then every node  $M$  in the subtree rooted at  $N.\text{LLINK}$  has the property that  $M.\text{YVAL} < N.\text{YVAL}$  and every node  $P$  in the subtree rooted at  $N.\text{RLINK}$  has the property that  $P.\text{YVAL} \geq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

# Example 2-d Trees



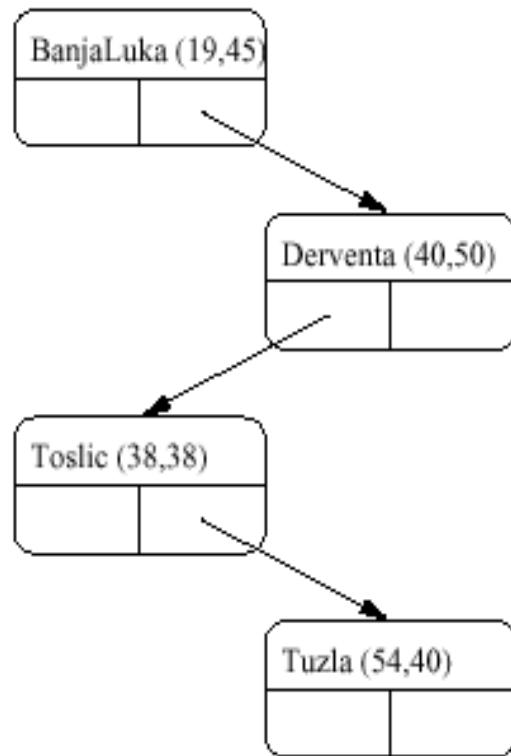
(a)

(b)

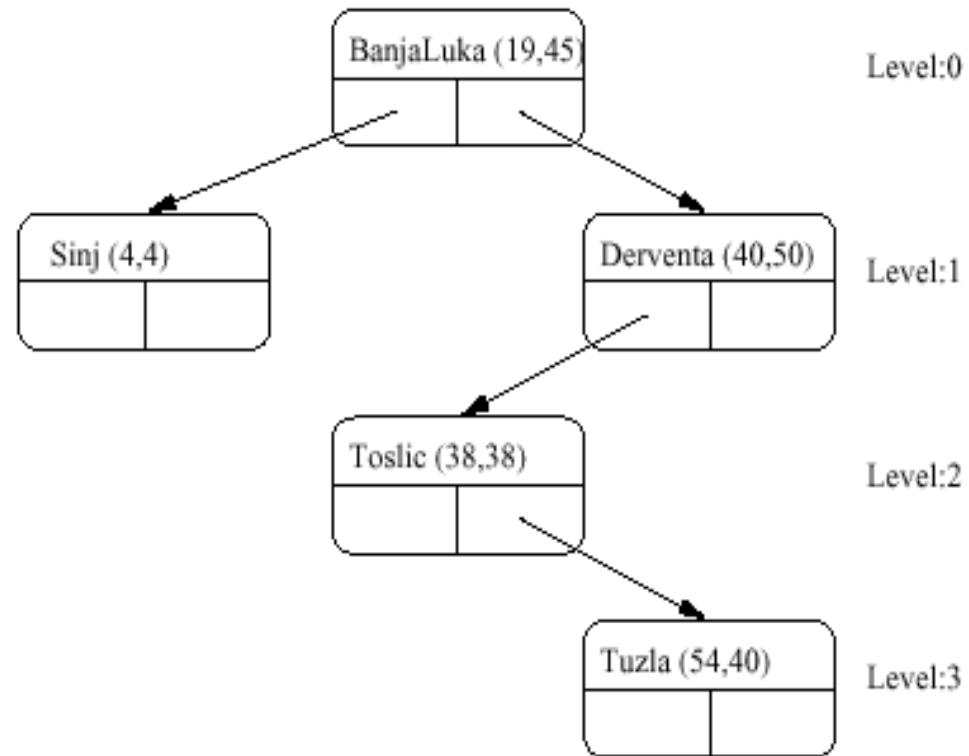


(c)

# Example 2-d Trees



(d)



(e)

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.

# Insertion/Search in 2-d Trees

- 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



# Example of Insertion

Suppose we wish to insert the following points.

City	(XVAL,YVAL)
Banja Luka	(19,45)
Derventa	(40,50)
Toslic	(38,38)
Tuzla	(54,35)
Sinj	(4,4)

# Example of Insertion



(a) Splitting of region by Banja Luka



### (b) Splitting of region by Derventa

# Example of Insertion



(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.

# Deletion in 2-d Trees

- Otherwise, either the subtree rooted at  $N.LLINK$  (which we will denote by  $T_l$ ) 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  $i \in \{l, r\}$ .  
**(Step 2)** Replace all of  $N$ 's non-link fields by those of  $R$ .  
**(Step 3)** Recursively delete  $R$  from  $T_i$ .

# Deletion in 2-d Trees

- The above recursion is guaranteed to terminate as  $T_i$  for  $i \in \{l, r\}$  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  $T_i$  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.

# Finding Candidate Replacement Nodes for Deletion

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

# Finding Candidate Replacement Nodes for Deletion

- 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?).

# Finding Candidate Replacement Nodes for Deletion

- In this case, find the node  $R'$  in  $T_1$  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.

# Range Queries in 2-d Trees

- 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 horizontal and a vertical line through the point (N.XVAL, N.YVAL).

# Point Quadtrees

- 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.

# Point Quadtrees

- Node structure in a point quadtree:

**qtnodetype = record**

INFO: **infotype**;

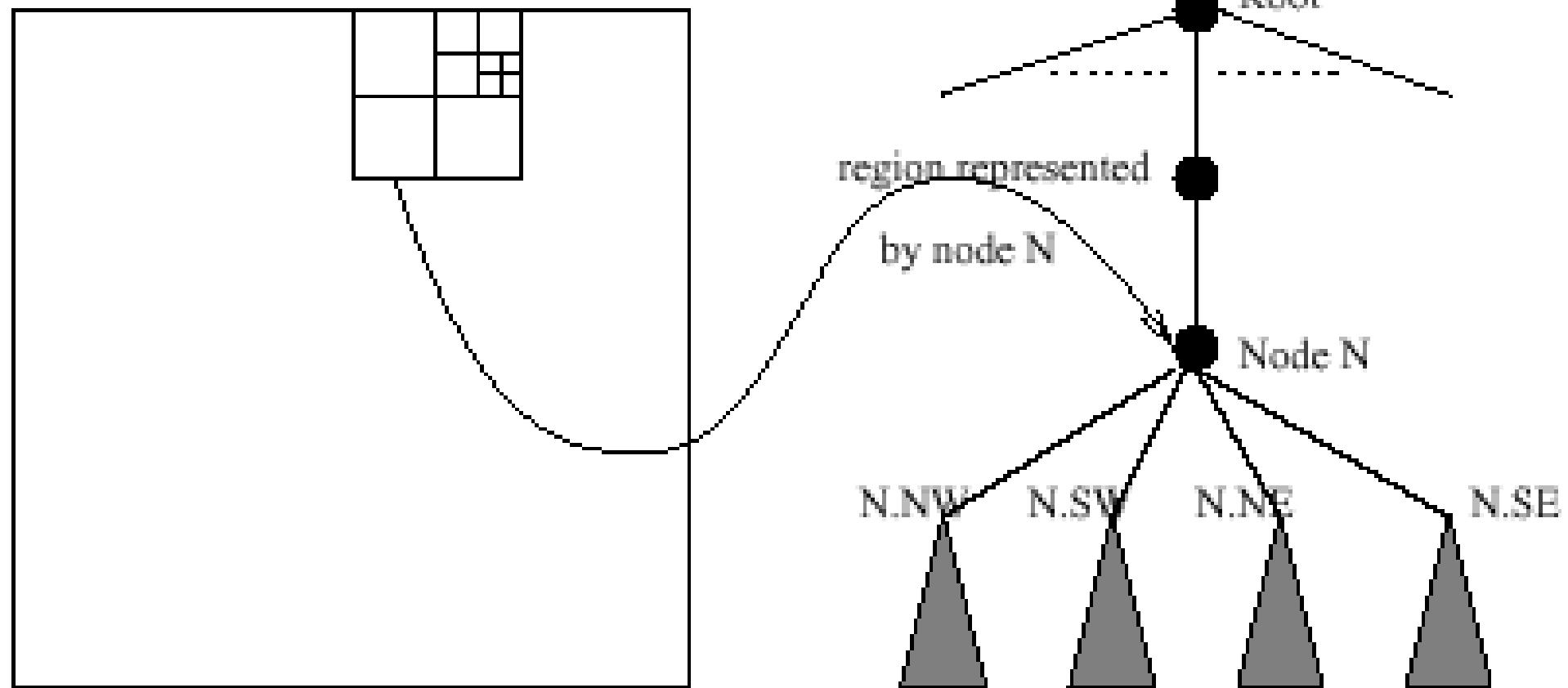
XVAL: **real**;

YVAL: **real**;

NW,SW,NE,SE:  $\uparrow$ **qtnodetype**

**end**

# Nodes in Point Quadtrees Implicitly Represent Regions



# Insertion into Point Quadtrees

City	(XVAL,YVAL)
Banja Luka	(19,45)
Derventa	(40,50)
Toslic	(38,38)
Tuzla	(54,35)
Sinj	(4,4)

# Insertion into Point Quadtrees



(a) Splitting of region by Banja Luka



(b) Splitting of region by Derventa

# Insertion into Point Quadtrees



(c) Splitting of region by Tuzla

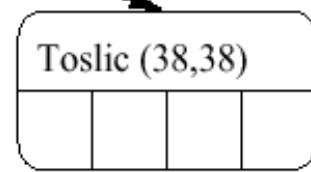
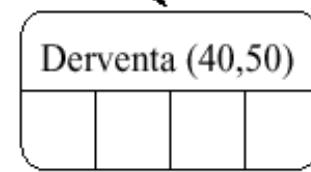
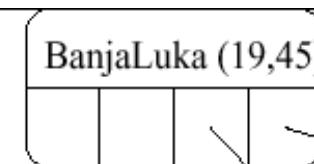
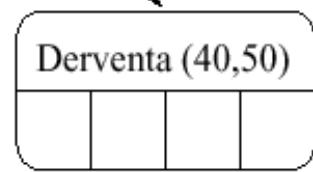
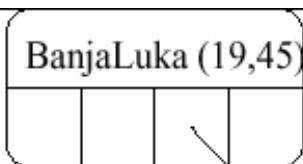
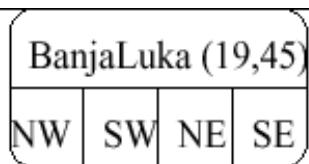


(d) Splitting of region by Jasenac

# Insertion into Point Quadtrees



# Insertion into Point Quadtrees

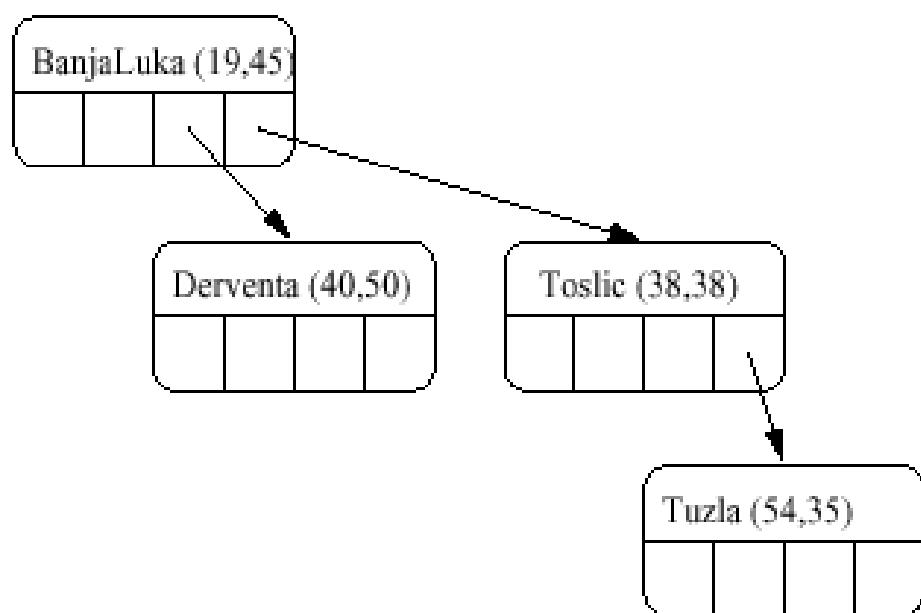


(a)

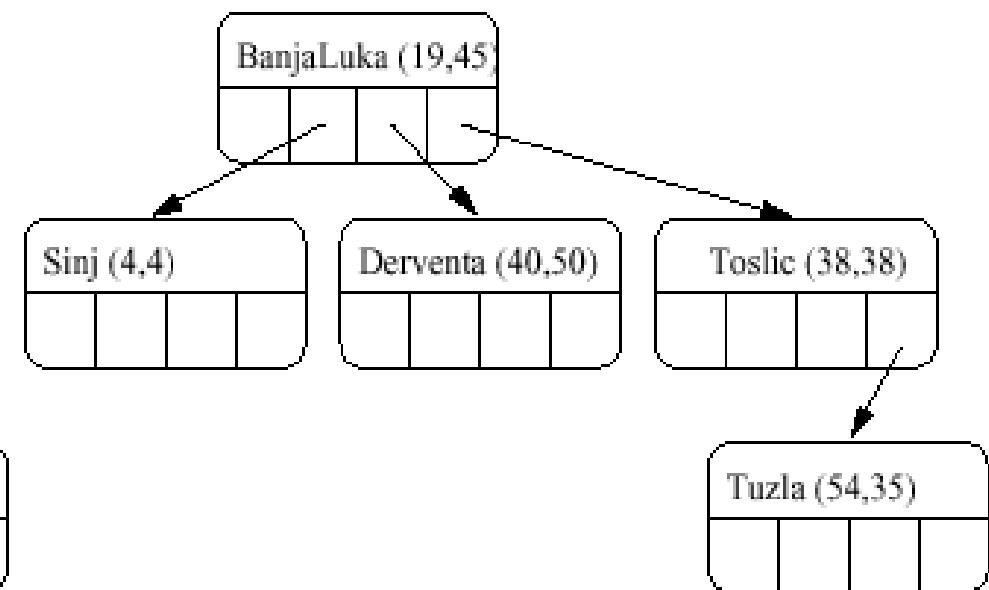
(b)

(c)

# Insertion into Point Quadtrees



(d)



(e)

# 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.

# Expanded Node Type

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

**qtnodetype** = record

  INFO: **infotype**;

  XVAL,YVAL: **real**;

  XLB,YLB,XUB,YUB: **real**  $\cup \{-\infty, +\infty\}$

  NW,SW,NE,SE:  $\uparrow$  **qtnodetype**

end

# Expanded Node Type

- When inserting a node N into the tree T, we need to ensure that:
  - If N is the root of tree T, then  $N.XLB = -\infty$ ,  $N.YLB = -\infty$ ,  $N.XUN = +\infty$ ,  $N.YUB = +\infty$ .
  - 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  $h = (Y.YUB - Y.YLB)$ .

# Expanded Node Type

Case	N.XLB	N.XUB	N.YLB	N.YUB
N=P.NW	P.XLB	P.XLB + $w \times 0.5$	P.YLB + $h \times 0.5$	P.YUB
N=P.SW	P.XLB	P.XLB + $w \times 0.5$	P.YLB	P.YLB + $h \times 0.5$
N=P.NE	P.XLB + $w \times 0.5$	P.XUB	P.YLB + $h \times 0.5$	P.YUB
N=P.SE	P.XLB + $w \times 0.5$	P.XUB	P.YLB	P.YLB + $h \times 0.5$

# 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  $N.NW, N.SW, N.NE, N.SE$ ) such that:
  - every other node  $R_{-}$  in  $N.NW$  is to the north west of  $R$ ,
  - every other node  $R_{-}$  in  $N.SW$  is to the south west of  $R$ ,
  - every other node  $R_{-}$  in  $N.NE$  is to the north east of  $R$  and
  - every other node  $R_{-}$  in  $N.SE$  is to the south east of  $R$ .

# Deletion in Point Quadtrees, Continued

- 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 Derven

# Deletion of Banja Luka



(c) Splitting of region by Toslic



(d) Splitting of region by Tuzla

# Deletion of Banja Luka



# Impossibility of finding Replacement Candidates



# Impossibility of finding Replacement Candidates

- 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.

# Range Searches in Point Quadtrees

```
proc RangeQueryPointQuadtree
  (T:newqtnodetype, C:circle);
  1. If region(T) ∩ C = Ø; then Halt
  2. else
    (a) If (T.XVAL, T.YVAL) ∈ C then print (T.XVAL, T.YVAL);
    (b) RangeQueryPointQuadtree(T.NW,C);
    (c) RangeQueryPointQuadtree(T.SW,C);
    (d) RangeQueryPointQuadtree(T.NE,C);
    (e) RangeQueryPointQuadtree(T.SE,C);
end proc
```

# The MX-Quadtree

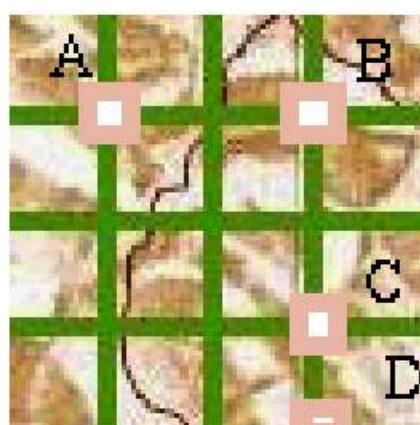
- For both 2-d trees as well as point quadtrees, the “shape” of the tree depends upon the order in which objects are inserted into the tree.
- In addition, both 2-d trees and point quadtrees split regions into 2 (for 2-trees) or 4 (for point quadtrees) sub-regions -- however, the split may be uneven depending upon exactly where the point (`N.XVAL`, `N.YVAL`) is located inside the region represented by node `N`.

# The MX-Quadtree

- 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.
- Ex:



# The MX-Quadtree

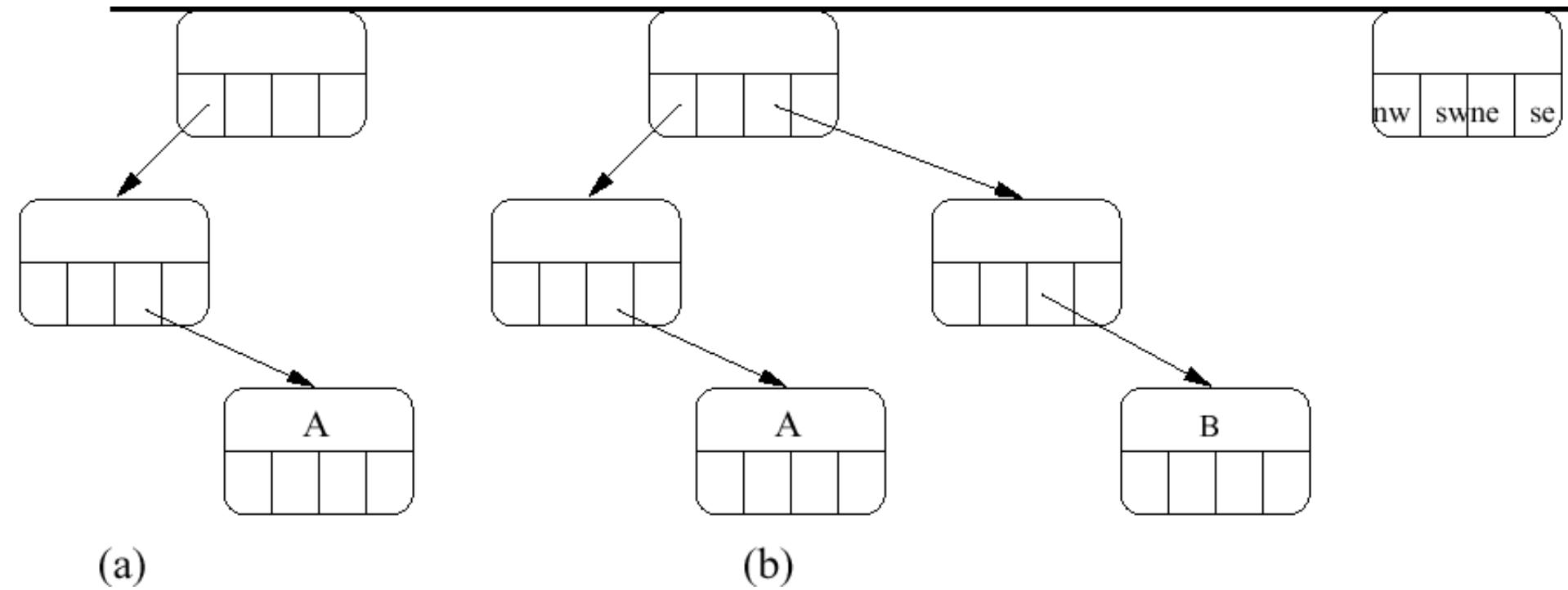
- Node Structure: Exactly the same as for point quadtrees, except that the root of an MX-quadtree represents the region specified by  $XLB= 0$ ,  $XUB= 2^k$ ,  $YLB= 0$ ,  $YUB=2^k$ .
- When a region gets “split”, it gets split down the middle.
- Thus, if  $N$  is a node, then the regions represented by the four children of  $N$  are described by the following table.

# The MX-Quadtree

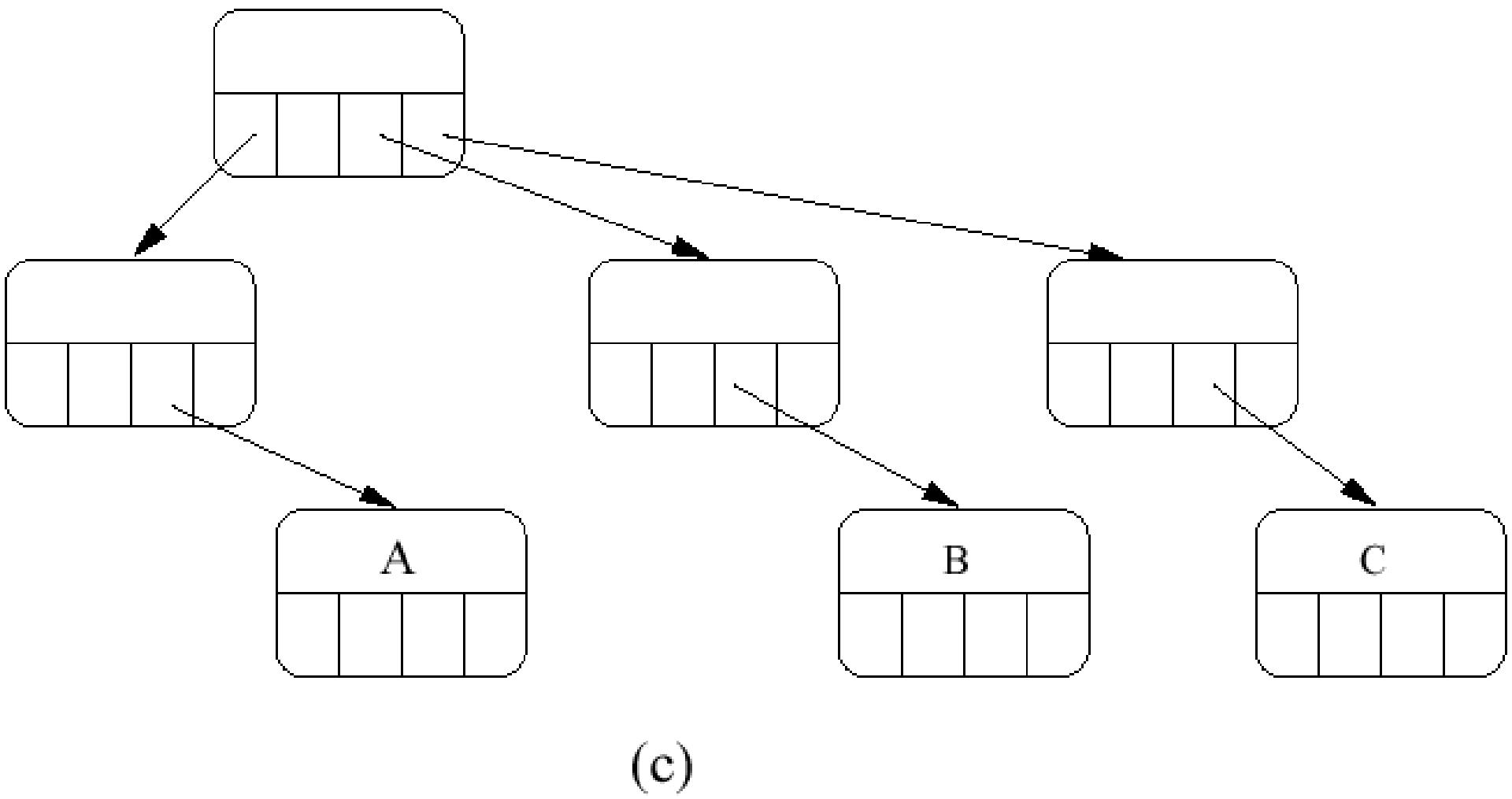
Child	XLB	XUB	YLB	YUB
NW	$N.XLB$	$N.XLB + \frac{w}{-}$	$N.YLB + \frac{w}{-}$	$N.YLB + w$
SW	$N.XLB$	$N.XLB + \frac{w}{-}$	$N.YLB$	$N.YLB + \frac{w}{-}$
NE	$N.XLB + \frac{w}{-}$	$N.XLB + \frac{w}{-}$	$N.YLB + \frac{w}{-}$	$N.YLB + \frac{w}{-}$
SE	$N.XLB + \frac{w}{-}$	$N.XLB + w$	$N.YLB$	$N.YLB + \frac{w}{-}$

Here,  $w$  denotes the width of the region represented by  $N$ .

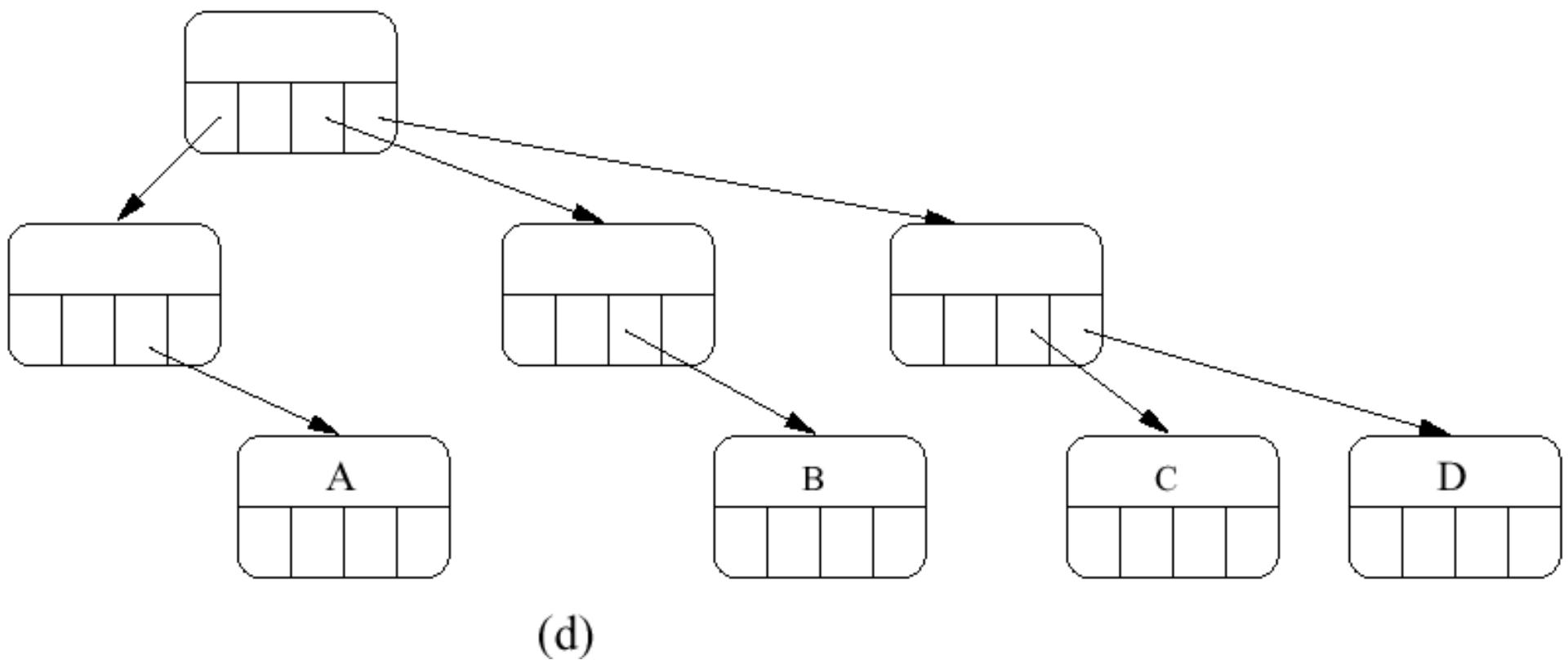
# Insertion in MX-Quadtrees



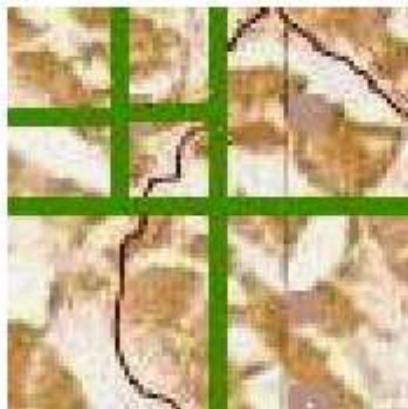
# Insertion in MX-Quadtrees



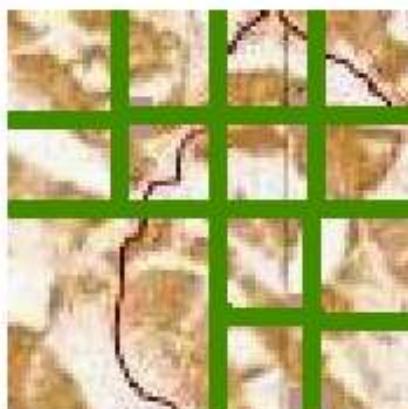
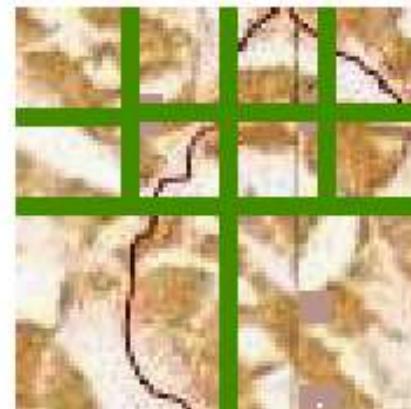
# Insertion in MX-Quadtrees



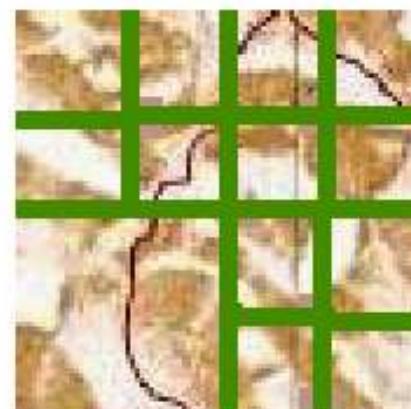
# Insertion in MX-Quadtrees



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.

# Deletion in MX-Quadtrees

- This can be done as follows.
  - First, we set the appropriate link of  $N$ 's parent to  $\text{NIL}$ .
  - We then check if all the four link fields of  $M$  are  $\text{NIL}$ .
  - 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  $dir1$  such that  $P.dir1 = M$ . We then set  $P.dir1 = \text{NIL}$  and then (as before) check to see if  $P$ 's four link fields are all  $\text{NIL}$ .
  - if 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 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 non-leaf R-tree node contains a set of at most  $K$  rectangles and at least  $[K/2]$  rectangles (with the possible exception of the root).
- Intuitively, this says that each non-leaf node in the R-tree, with the exception of the root, must be at least “half” full.

# R-Trees

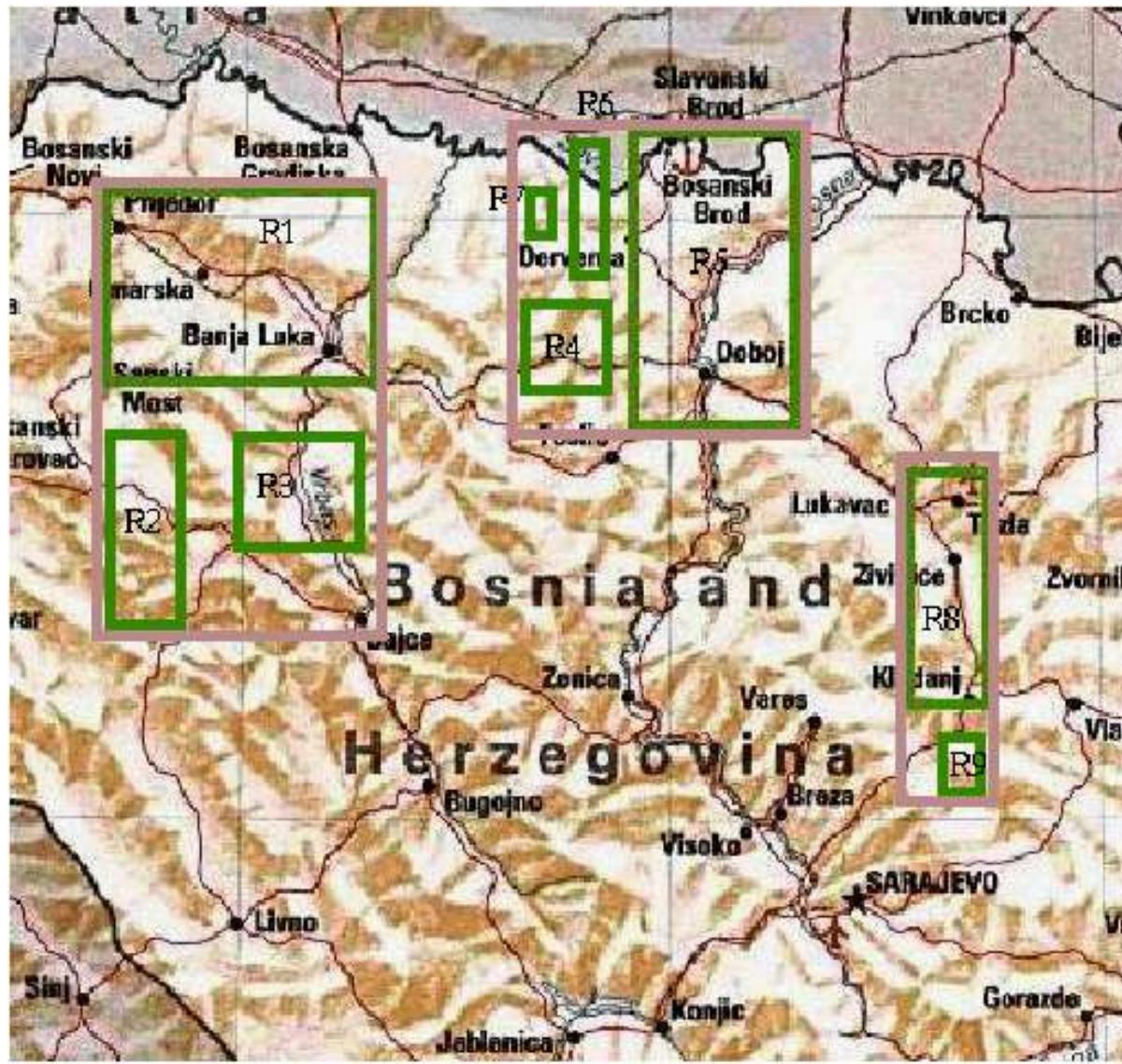
- This feature makes R-trees appropriate for disk based retrieval because each disk access brings back a page containing several (i.e. at least  $K/2$  rectangles).

# R-Trees

R-trees manipulate two kinds of rectangles:

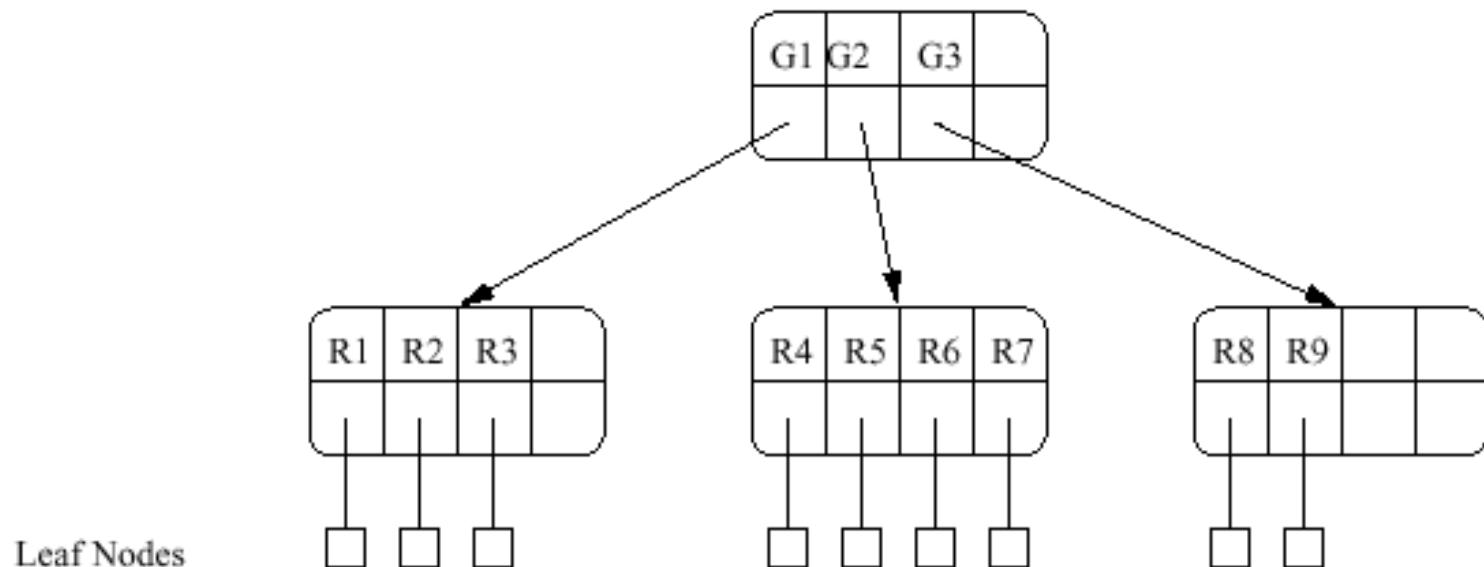
- “Real” rectangles (such as those shown in the map on the previous slide) or
- “Group” rectangles such as those shown below.

# R-Trees



# Example R-Tree

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



# Example R-Tree

R-tree nodes have the following structure:

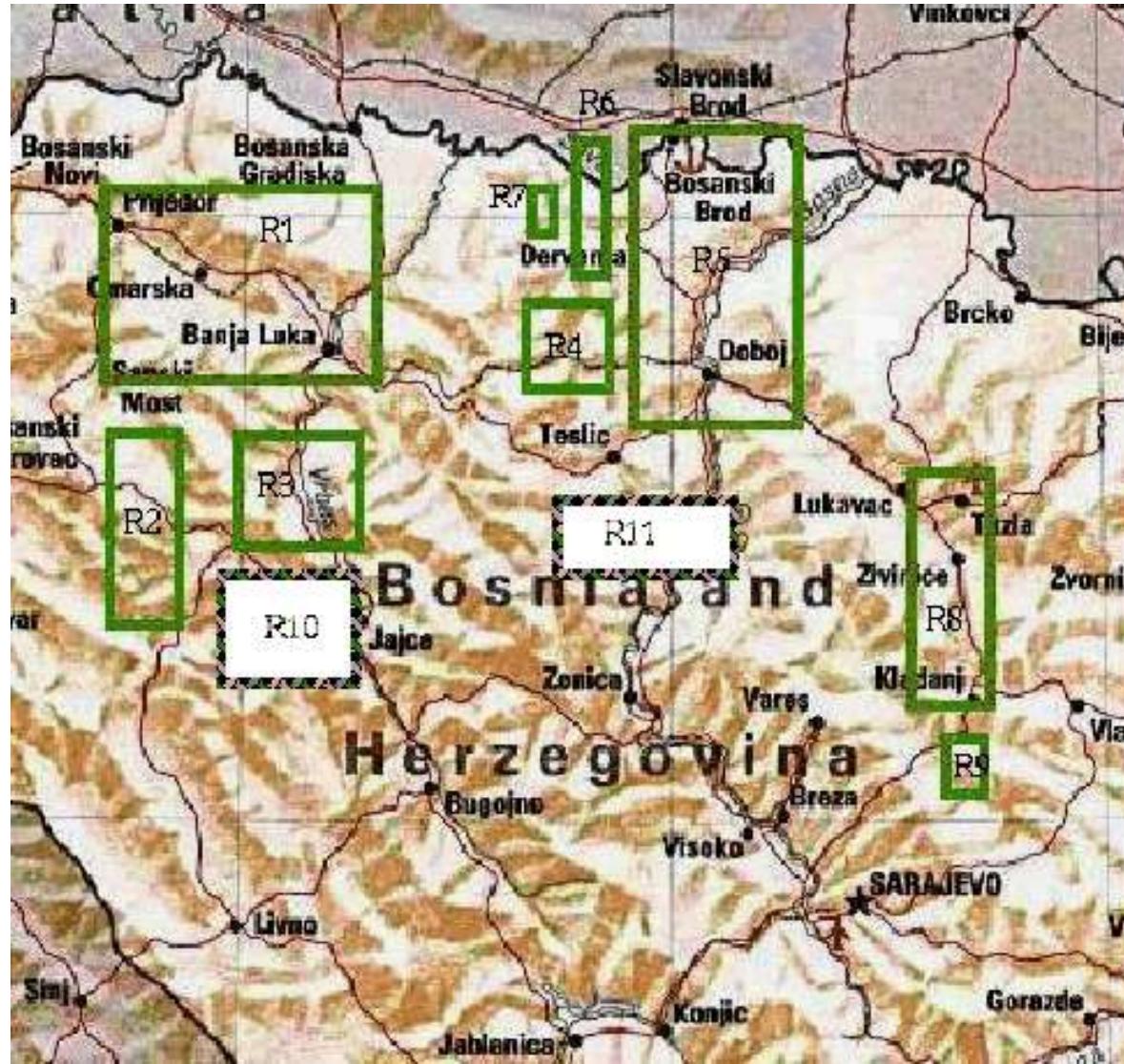
**rtnodetype = record**

$\text{Rec}_1, \dots, \text{Rec}_K$  : **rectangle**;

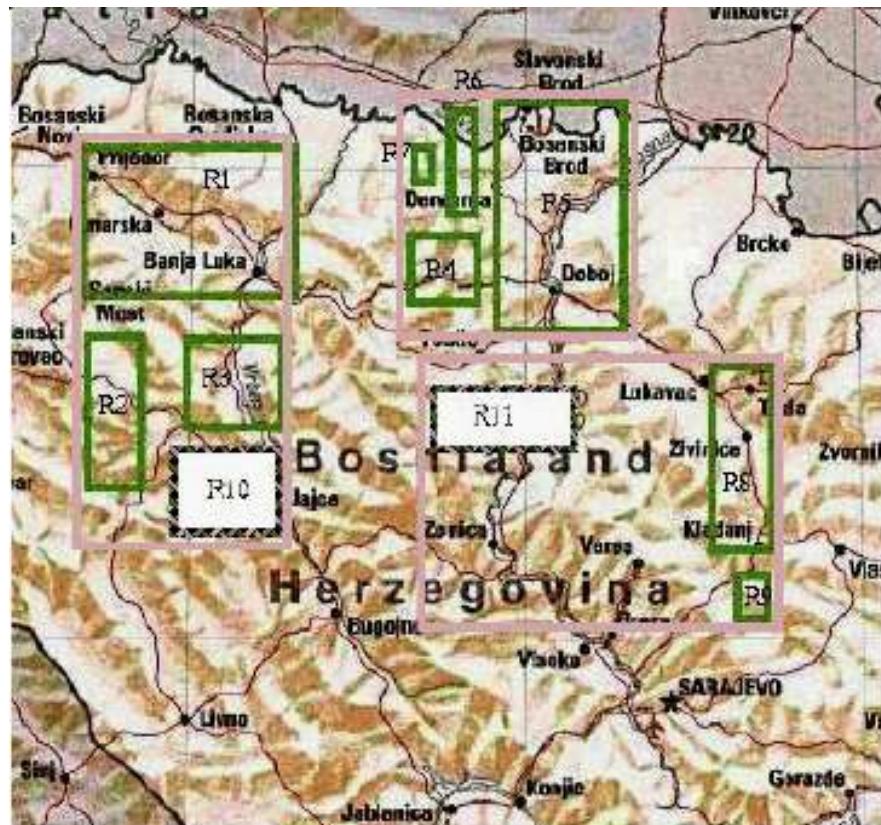
$P_1, \dots, P_K$  :  $\uparrow$ **rtnodetype**

end

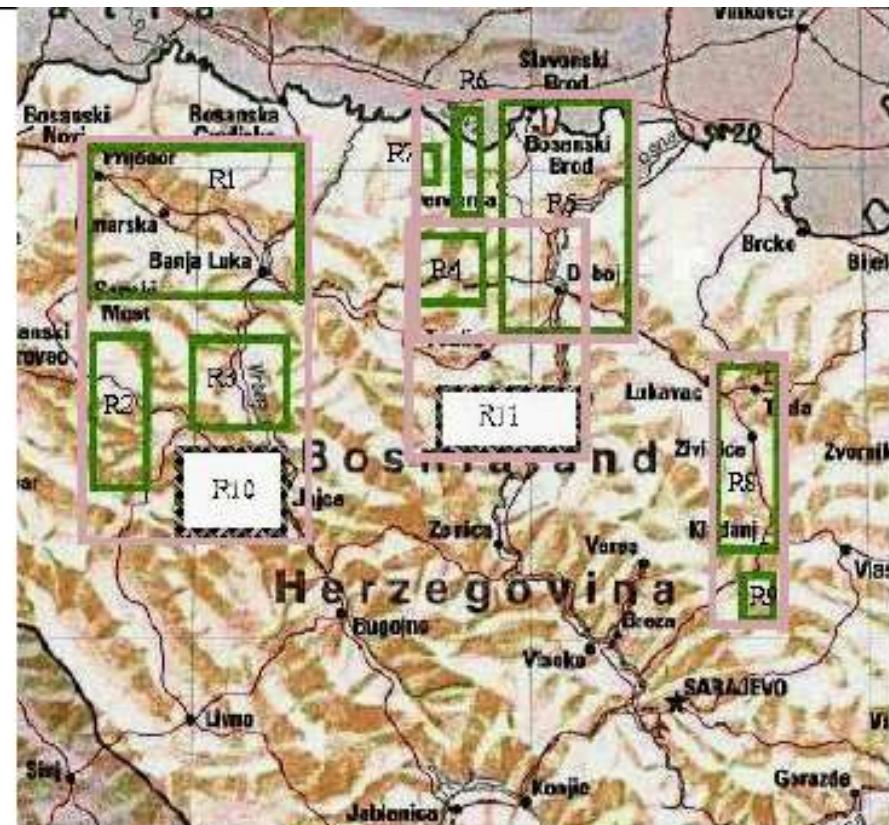
# Insertion into an R-Tree



# Insertion into an R-Tree

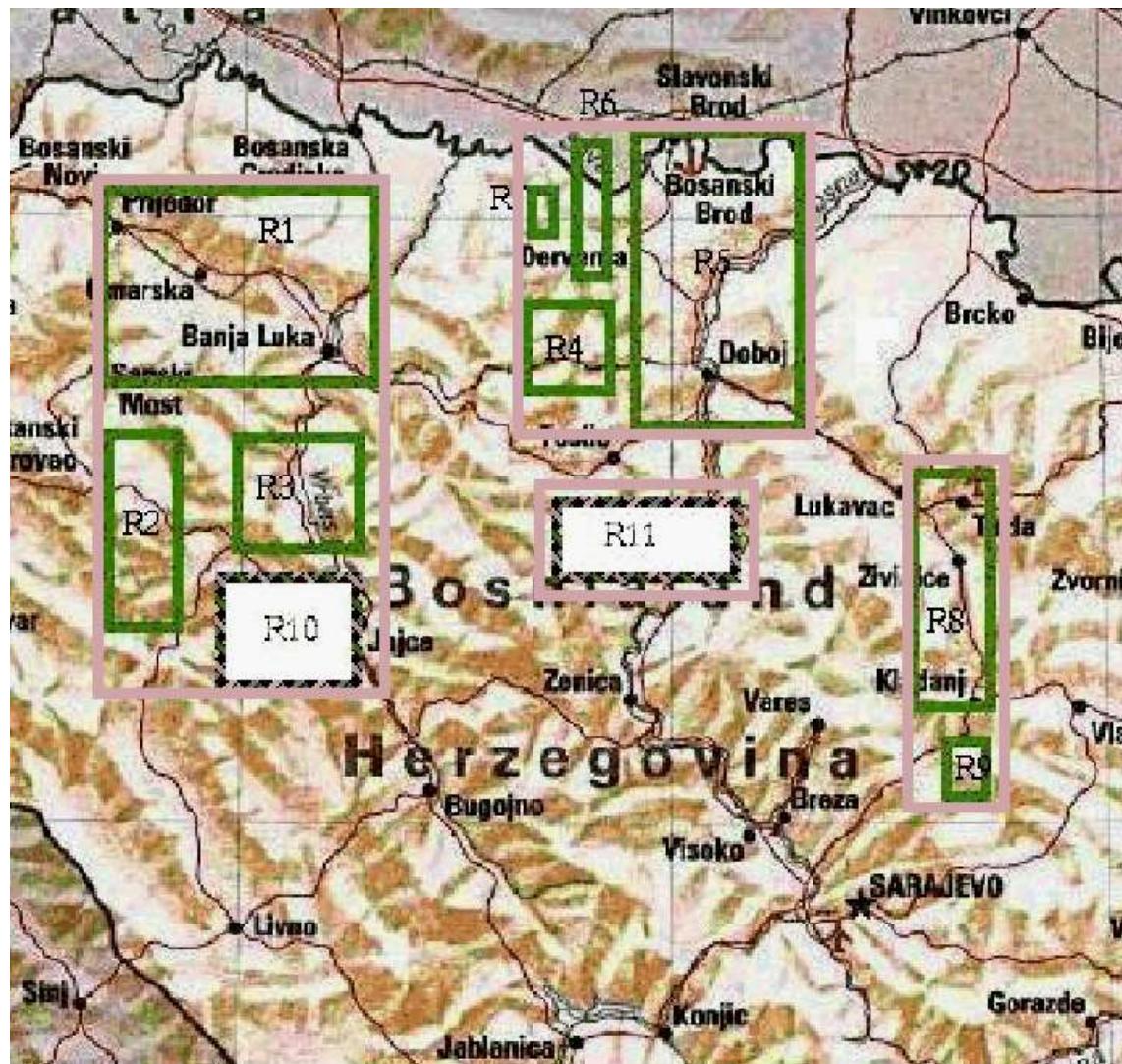


(a)



(b)

# An Incorrect Insertion into an R-Tree

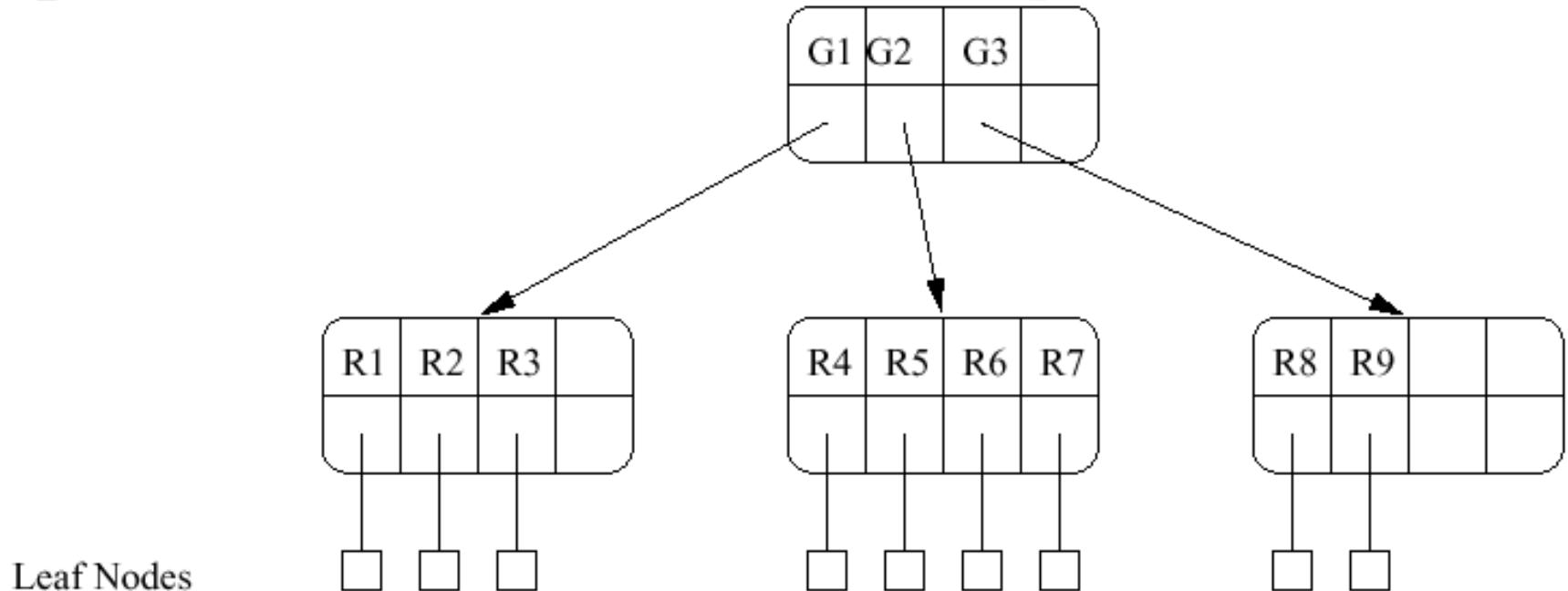


# Deletion in R-Trees

- 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  $[K/2]$  rectangles (real or group) in it.
- When we delete a rectangle from an R-tree, we must ensure that that node is not “under-full”.

# Deletion in R-Trees

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



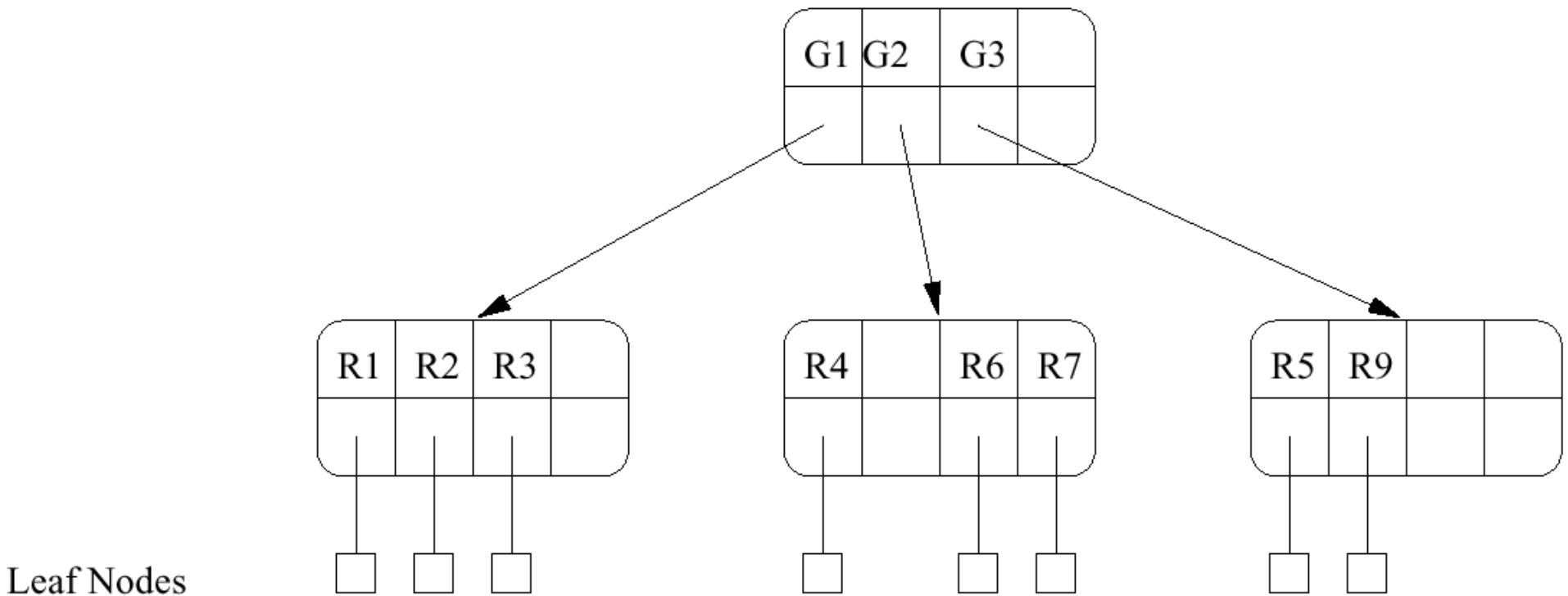
# Deletion in R-Trees

- 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:

Group	Rectangles
G1	R1,R2,R3
G2	R4,R6,R7
G3	R5,R8

# Deletion in R-Trees

- The new R-tree is:





Cairo University



# Geographic Information System And Spatial Databases

Course Code: IS443

## Chapter 3: Spatial Query Languages

- **Traditional (non-spatial) database management systems provide:**
  - Persistence across failures
  - Allows concurrent access to data
  - Scalability to search queries on very large datasets which do not fit inside main memories of computers
  - Efficient for non-spatial queries, but not for spatial queries
- **Non-spatial queries:**
  - List the names of all bookstore with more than ten thousand titles.
  - List the names of ten customers, in terms of sales, in the year 2001
- **Spatial Queries:**
  - List the names of all bookstores with ten miles of Minneapolis
  - List all customers who live in Tennessee and its adjoining states

# What is a SDBMS ?

- A SDBMS is a software module that
  - can work with an underlying DBMS
  - supports spatial data models, spatial abstract data types (ADTs) and a query language from which these ADTs are callable
  - supports spatial indexing, efficient algorithms for processing spatial operations, and domain specific rules for query optimization
- Example: Oracle Spatial data cartridge, ESRI SDE
  - can work with Oracle 8i DBMS
  - Has spatial data types (e.g. polygon), operations (e.g. overlap) callable from SQL3 query language
  - Has spatial indices, e.g. R-trees

# SDBMS Example

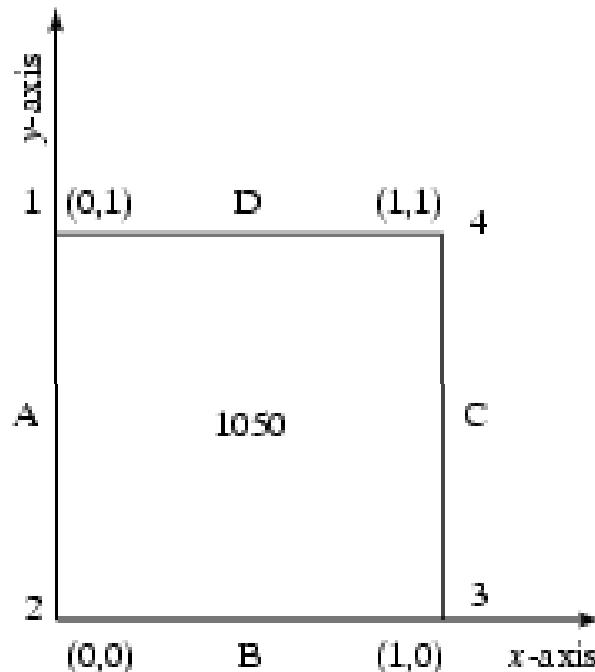
- Consider a spatial dataset with:
  - County boundary (dashed white line)
  - Census block - name, area, population, boundary (dark line)
  - Water bodies (dark polygons)
  - Satellite Imagery (gray scale pixels)
- Storage in a SDBMS table:

```
create table census_blocks (  
    name  string,  
    area      float,  
    population   number,  
    boundary polygon );
```



# Modeling Spatial Data in Traditional DBMS

- A row in the table `census_blocks`
- Question: Is Polyline datatype supported in DBMS?



Census\_blocks

Name	Area	Population	Boundary
1050	1	1839	Polyline((0,0),(0,1),(1,1),(1,0))

# Spatial Query Language

- Spatial query language

- Spatial data types, e.g. point, linestring, polygon, ...
- Spatial operations, e.g. overlap, distance, nearest neighbor, ...
- Callable from a query language (e.g. SQL3) of underlying DBMS

```
SELECT S.name  
FROM Senator S  
WHERE S.district.Area() > 300
```

- Standards

- SQL3 (a.k.a. SQL 1999) is a standard for query languages
- OGIS is a standard for spatial data types and operators
- Both standards enjoy wide support in industry

# Multi-scan Query Example

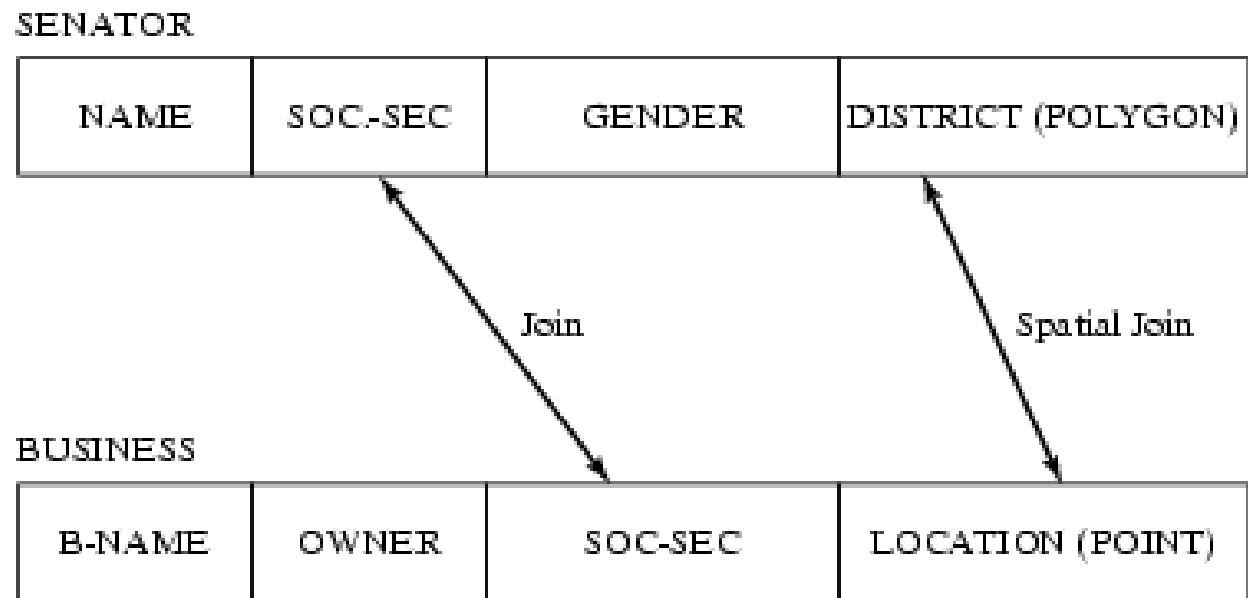
- Spatial join example

```
SELECT S.name  
FROM Senator S, Business B
```

WHERE S.district.Area() > 300 AND Within(B.location, S.district)

- Non-Spatial Join example

```
SELECT S.name      FROM Senator S, Business B  
WHERE S.soc-sec = B.soc-sec AND S.gender = 'Female'
```

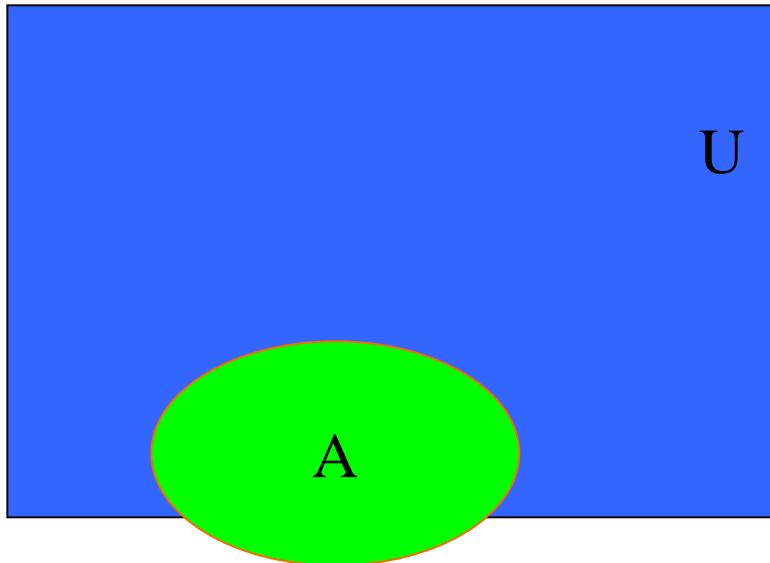


# Topological Relationships

- Topological Relationships
  - invariant under elastic deformation (without tear, merge).
  - Two countries which touch each other in a planar paper map will continue to do so in spherical globe maps.
- Topology is the study of topological relationships
- Example queries with topological operations
  - What is the topological relationship between two objects A and B ?
  - Find all objects which have a given topological relationship to object A ?

# Topological Concepts

- Interior, boundary, exterior
  - Let  $A$  be an object in a “Universe”  $U$ .



Green is A interior ( $A^\circ$ )

Red is boundary of A ( $\partial A$ )

Blue –(Green + Red) is  
A exterior ( $A^-$ )

- Question: Define Interior, boundary, exterior on curves and points.

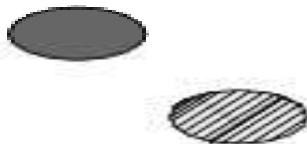
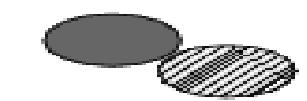
# Nine-Intersection Model of Topological Relationships

- Many topological Relationship between A and B can be
  - specified using 9 intersection model
  - Examples on next slide
- Nine intersections
  - intersections between interior, boundary, exterior of A, B
  - A and B are spatial objects in a two dimensional plane.
  - Can be arranged as a 3 by 3 matrix
  - Matrix element take a value of 0 (false) or 1 (true).
- Q? Determine the number of many distinct 3 by 3 boolean matrices .

$$\Gamma_9(A, B) = \begin{pmatrix} A^\circ \cap B^\circ & A^\circ \cap \partial B & A^\circ \cap B^- \\ \partial A \cap B^\circ & \partial A \cap \partial B & \partial A \cap B^- \\ A^- \cap B^\circ & A^- \cap \partial B & A^- \cap B^- \end{pmatrix}$$

# Specifying topological operation in 9-Intersection Model

9 intersection matrices for a few topological operations

			
$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ disjoint	$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ contains	$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ inside	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ equal
			
$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ meet	$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ covers	$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ coveredBy	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ overlap

Question: Can this model specify topological operation between a polygon and a curve?  
220

# Extending ER with Spatial Concepts

- Motivation

- ER Model is based on discrete sets with no implicit relationships
- Spatial data comes from a continuous set with implicit relationships
- Any pair of spatial entities has relationships like distance, direction, ...

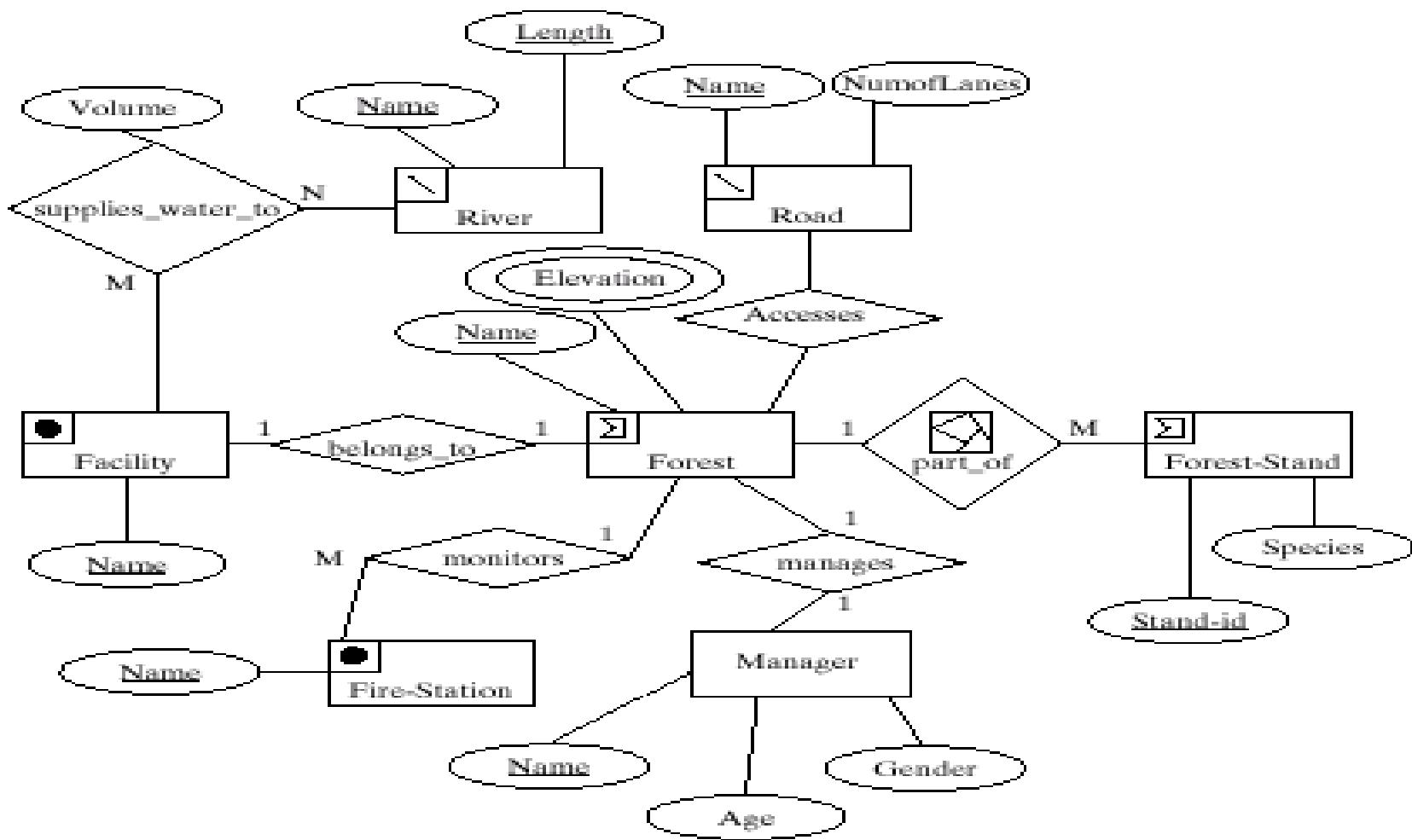
- Explicitly drawing all spatial relationship

- clutters ER diagram
- generates additional tables in relational schema
- Misses implicit constraints in spatial relationships (e.g. partition)

- Pictograms

- Label spatial entities along with their spatial data types
- Allows inference of spatial relationships and constraints
- Reduces clutter in ER diagram and relational schema
- Example: Fig. 2.7 (next slide) is simpler than Fig. 2.4

# ER Diagram with Pictograms: An Example



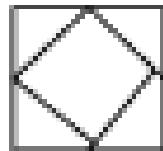
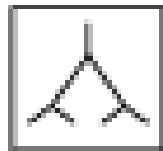
# Specifying Pictograms

- Grammar based approach

- Rewrite rule
  - like English syntax diagrams

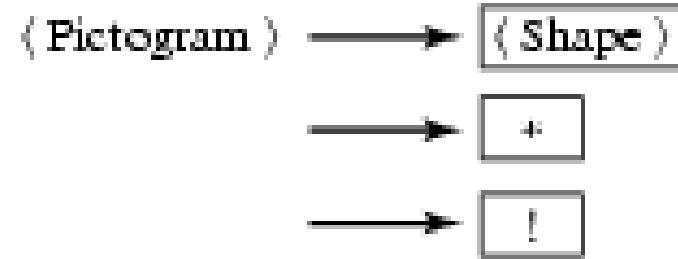
- Classes of pictograms

- Entity pictograms
  - basic: point, line, polygon
  - collection of basic
  - ...
- Relationship pictograms
  - partition, network



Part\_of(Network)   Part\_of(Partition)

Pictograms for Relationships



*Grammar (for Pictogram)*



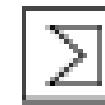
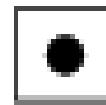
*Grammar (for Shape)*

# Entity Pictograms: Basic shapes, Collections

{ Basic Shape } → ●

→ /

→  $\Sigma$



Point

Line

Polygon

*Grammar (for Basic Shape)*

*Pictograms for Basic Shapes*

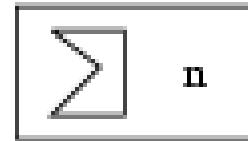
{ Cardinality } → 0,1

→ 1

→ 1,n

→ 0,n

→ n



22 *Grammar (for Cardinality)*

*Pictograms Multishapes  
(using cardinality)*

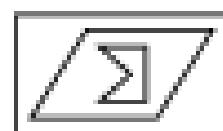
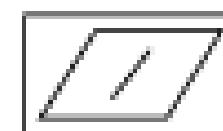
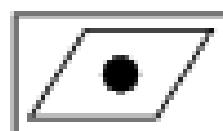
# Entity Pictograms: Derived and Alternate Shapes

- Derived shape example is city center point from boundary polygon
- Alternate shape example: A road is represented as a polygon for construction
  - or as a line for navigation

**( Derived Shape )**



**( Basic Shape )**



*Grammar (for Derived Shape)*

**( Alternate Shape )**

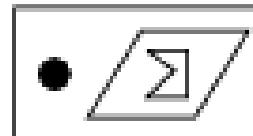
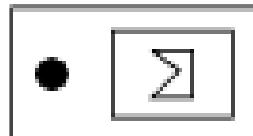


**( Basic Shape ) ( Derived Shape )**



**( Basic Shape ) ( Basic Shape )**

*Grammar (for Alternate Shape)*

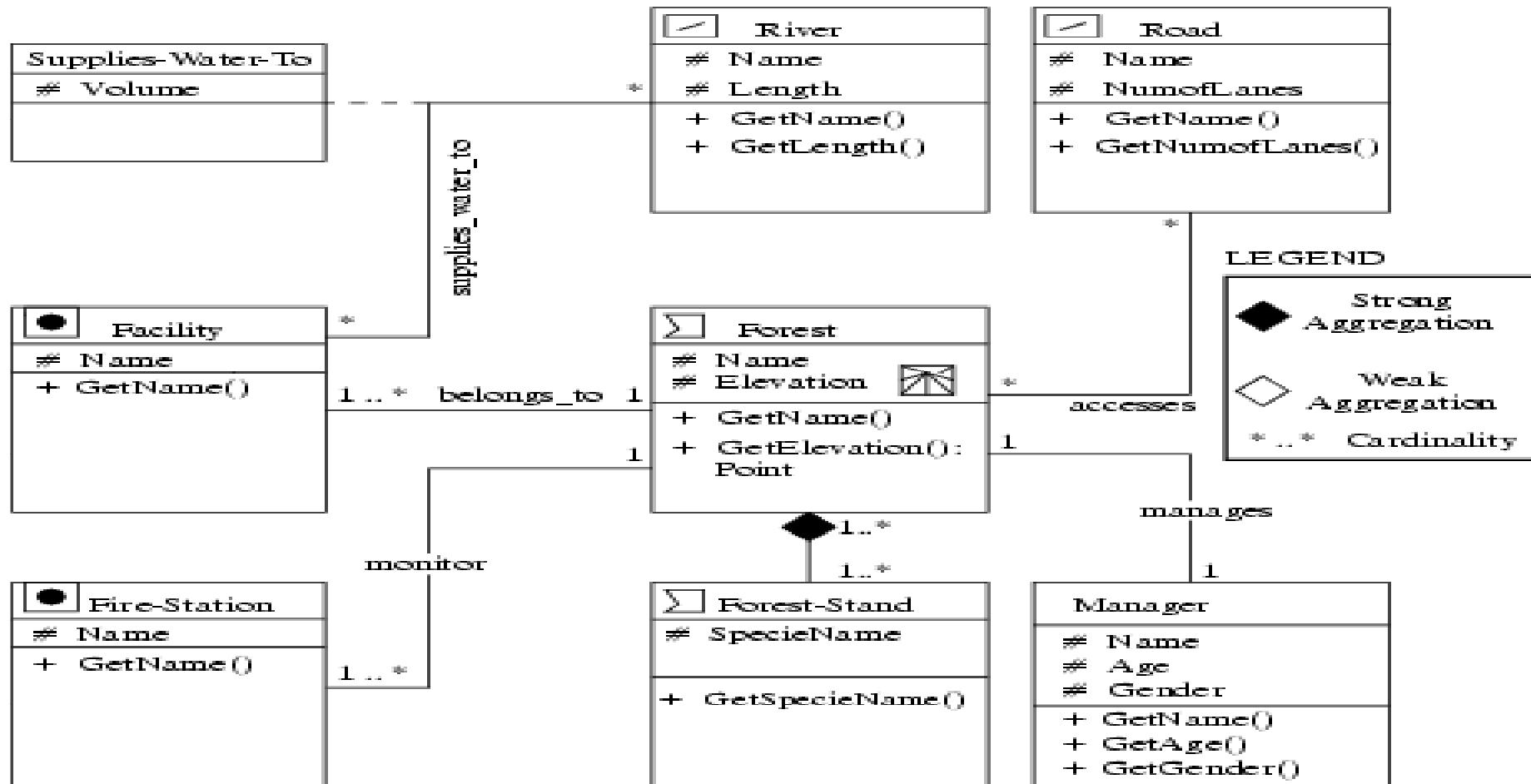


*Pictograms for Alternate Shapes*

# Conceptual Data Modeling with UML

- Motivation
  - ER Model does not allow user defined operations
  - Object oriented software development uses UML
  - UML stands for Unified Modeling Language
  - It is a standard consisting of several diagrams
    - class diagrams are most relevant for data modeling
- UML class diagrams concepts
  - Attributes are simple or composite properties
  - Methods represent operations, functions and procedures
  - Class is a collection of attributes and methods
  - Relationship relate classes

# UML Class Diagram with Pictograms: Example

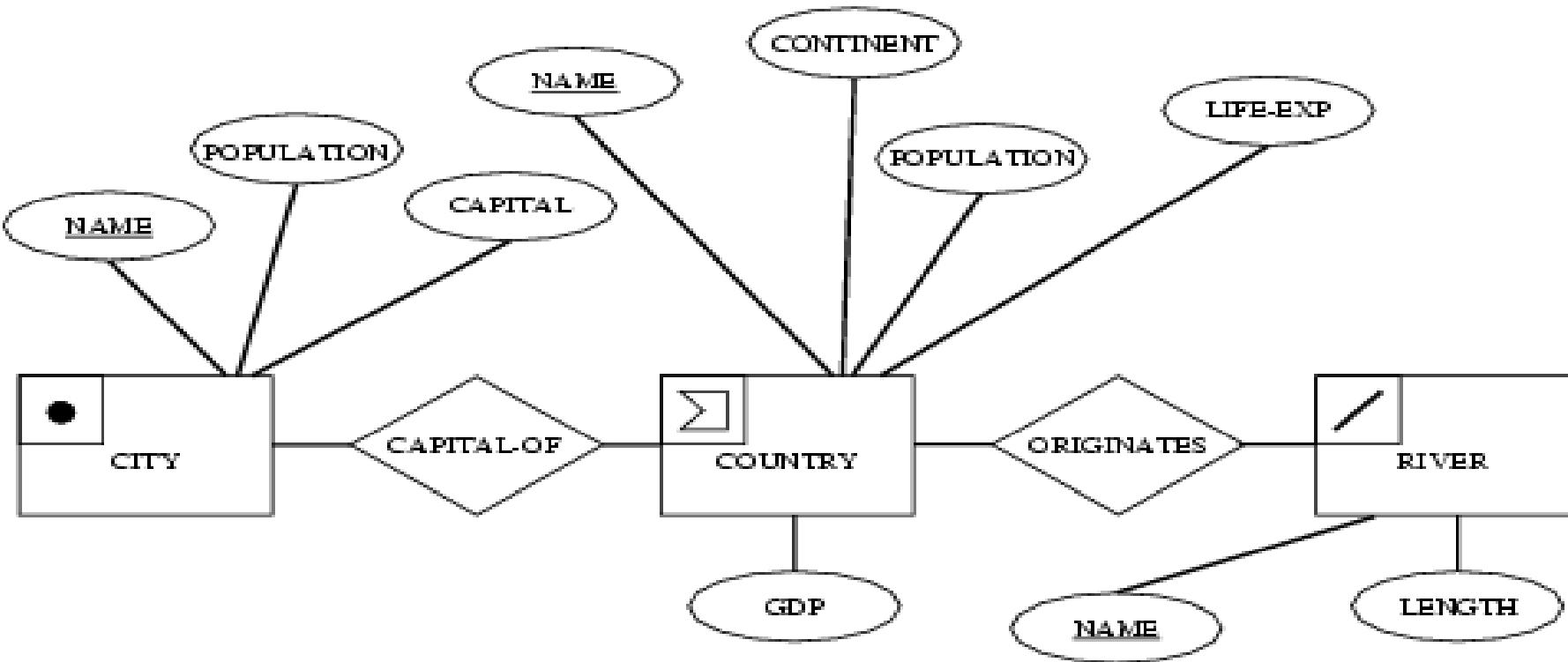


# Comparing UML Class Diagrams to ER Diagrams

- Concepts in UML class diagram vs. those in ER diagrams
  - Class without methods is an Entity
  - Attributes are common in both models
  - UML does not have key attributes and integrity constraints
    - ERD does not have methods
    - Relationships properties are richer in ERDs
    - Entities in ER diagram relate to datasets, but UML class diagram
      - can contain classes which have little to do with data

# An Example World Database

- Purpose: Use an example database to learn query language SQL
- Conceptual Model
  - 3 Entities: Country, City, River
  - 2 Relationships: capital-of, originates-in
  - Attributes listed in Figure 3.1



# An Example Database - Logical Model

- 3 Relations

**Country**(*Name, Cont, Pop, GDP, Life-Exp, Shape*)

**City**(*Name, Country, Pop, Capital, Shape*)

**River**(*Name, Origin, Length, Shape*)

- Keys

- Primary keys are *Country.Name, City.Name, River.Name*

- Foreign keys are *River.Origin, City.Country*

- Data for 3 tables

- Shown on next slide

# World database data tables

COUNTRY	Name	Cont	Pop (millions)	GDP (billions)	Life-Exp	Shape
	Canada	NAM	30.1	658.0	77.08	Polygonid-1
	Mexico	NAM	107.5	694.3	69.36	Polygonid-2
	Brazil	SAM	183.3	1004.0	65.60	Polygonid-3
	Cuba	NAM	11.7	16.9	75.95	Polygonid-4
	USA	NAM	270.0	8003.0	75.75	Polygonid-5
	Argentina	SAM	36.3	348.2	70.75	Polygonid-6

(a) Country

CITY	Name	Country	Pop (millions)	Capital	Shape
	Havana	Cuba	2.1	Y	Pointid-1
	Washington, D.C.	USA	3.2	Y	Pointid-2
	Monterrey	Mexico	2.0	N	Pointid-3
	Toronto	Canada	3.4	N	Pointid-4
	Brasilia	Brazil	1.5	Y	Pointid-5
	Rosario	Argentina	1.1	N	Pointid-6
	Ottawa	Canada	0.8	Y	Pointid-7
	Mexico City	Mexico	14.1	Y	Pointid-8
	Buenos Aires	Argentina	10.75	Y	Pointid-9

(b) City

RIVER	Name	Origin	Length (kilometers)	Shape
	Rio Parana	Brazil	2600	LineStringid-1
	St. Lawrence	USA	1200	LineStringid-2
	Rio Grande	USA	3000	LineStringid-3
	Mississippi	USA	6000	LineStringid-4

(c) River

# Creating Tables in SQL

- Table definition
  - “CREATE TABLE” statement
  - Specifies **table name, attribute names and data types**
  - Create a table **with no rows**.
  - See an example at the bottom
- Related statements
  - ALTER TABLE statement modifies table schema if needed
  - DROP TABLE statement removes an empty table

```
CREATE TABLE River(  
    Name      varchar(30) ,  
    Origin    varchar(30) ,  
    Length    number,  
    Shape     LineString );
```

# Populating Tables in SQL

- Adding a row to an existing table

- “**INSERT INTO**” statement
  - Specifies table name, attribute names and values
  - Example:

```
INSERT INTO River(Name, Origin, Length) VALUES('Mississippi', 'USA', 6000)
```

- Related statements

- **SELECT statement with INTO clause** can insert multiple rows in a table
  - Bulk load, import commands also add multiple rows
  - **DELETE statement** removes rows
  - **UPDATE statement** can change values within selected rows

# Querying populated Tables in SQL

- SELECT statement
  - The commonly used statement to query data in one or more tables
  - Returns a relation (table) as result
  - Has many clauses
  - Can refer to many operators and functions
  - Allows nested queries which can be hard to understand
- Scope of our discussion
  - Learn enough SQL to appreciate spatial extensions
    - Observe example queries
  - Read and write simple SELECT statement
    - Understand frequently used clauses, e.g. SELECT, FROM, WHERE
    - Understand a few operators and function

# SELECT Statement- General Information

- Clauses
  - SELECT specifies desired columns
  - FROM specifies relevant tables
  - WHERE specifies qualifying conditions for rows
  - ORDER BY specifies sorting columns for results
  - GROUP BY, HAVING specifies aggregation and statistics
- Operators and functions
  - arithmetic operators, e.g. +, -, ...
  - comparison operators, e.g. =, <, >, BETWEEN, LIKE...
  - logical operators, e.g. AND, OR, NOT, EXISTS,
  - set operators, e.g. UNION, IN, ALL, ANY, ...
  - statistical functions, e.g. SUM, COUNT, ...
  - many other operators on strings, date, currency, ...

# SELECT Example 1.

- Simplest Query has SELECT and FROM clauses
  - Query: List all the cities and the country they belong to.

SELECT Name, Country

FROM CITY

Result



Name	Country
Havana	Cuba
Washington, D.C.	USA
Monterrey	México
Toronto	Canada
Brasilia	Brazil
Rosario	Argentina
Ottawa	Canada
Mexico City	México
Buenos Aires	Argentina

# SELECT Example 2.

- Commonly 3 clauses (SELECT, FROM, WHERE) are used
  - **Query:** List the names of the capital cities in the CITY table.

SELECT \*

FROM CITY

WHERE CAPITAL='Y'

Name	Country	Pop(millions)	Capital	Shape
Havana	Cuba	2.1	Y	Point
Washington, D.C.	USA	3.2	Y	Point
Brasilia	Brazil	1.5	Y	Point
Ottawa	Canada	0.8	Y	Point
Mexico City	Mexico	14.1	Y	Point
Buenos Aires	Argentina	10.75	Y	Point

Result →

# Query Example...Where clause

**Query:** List the attributes of countries in the Country relation **where the life-expectancy** is less than seventy years.

```
SELECT Co.Name,Co.Life-Exp  
FROM Country Co  
WHERE Co.Life-Exp <70
```

Note: use of alias 'Co' for Table 'Country'

Result →

Name	Life-exp
Mexico	69.36
Brazil	65.60

# Multi-table Query Examples

**Query:** List the capital cities and populations of countries whose GDP exceeds one trillion dollars.

Note: Tables City and Country **are joined by matching** “City.Country = Country.Name”. This simulates relational operator “join” discussed in 3.2

```
SELECT Ci.Name,Co.Pop  
FROM City Ci,Country Co  
WHERE Ci.Country =Co.Name  
AND Co.GDP >1000.0  
AND Ci.Capital='Y '
```

Ci.Name	Co.Pop
Brasilia	183.3
Washington, D.C.	270.0

# Multi-table Query Example

**Query:** What is the name and population of the capital city in the country where the St. Lawrence River originates?

```
SELECT Ci.Name, Ci.Pop  
FROM City Ci, Country Co, River R  
WHERE R.Origin =Co.Name  
AND Co.Name =Ci.Country  
AND R.Name ='St.Lawrence '  
AND Ci.Capital='Y '
```

Note: Three tables are joined together pair at a time. River.Origin is matched with Country.Name and City.Country is matched with Country.Name. The order of join is decided by query optimizer and does not affect the result.

# Query Examples...Aggregate Statistics

**Query:** What is the **average population** of the noncapital cities listed in the City table?

```
SELECT AVG(Ci.Pop)  
FROM City Ci  
WHERE Ci.Capital='N '
```

**Query:** For each continent, find the average GDP.

```
SELECT Co.Cont,Avg(Co.GDP)AS Continent-GDP  
FROM Country Co  
GROUP BY Co.Cont
```

# Query Example..Having clause, Nested queries

**Query:** For each country in which at least two rivers originate, find the length of the smallest river.

```
SELECT R.Origin, MIN(R.length) AS Min-length  
FROM River  
GROUP BY R.Origin  
HAVING COUNT(*) > 1
```

**Query:** List the countries whose GDP is greater than that of Canada.

## Using spatial operation in SELECT clause

**Query:** List the name, population, and area of each country listed in the Country table.

```
SELECT C.Name,C.Pop, Area(C.Shape)AS "Area"  
FROM Country C
```

Note: This query uses spatial operation, Area(). Note the use of **spatial operation in place of a column** in SELECT clause.

# Using spatial operator Distance

**Query:** List the GDP and the distance of a country's capital city to the equator for all countries.

```
SELECT Co.GDP, Distance(Point(0,Ci.Shape.y),Ci.Shape) AS  
"Distance"  
FROM Country Co,City Ci  
WHERE Co.Name = Ci.Country  
AND Ci.Capital = 'Y'
```

Co. Name	Co. GDP	Dist-to-Eq (in Km)
Havana	16.9	2562
Washington, D.C.	8003	4324
Brasilia	1004	1756
Ottawa	658	5005
Mexico City	694.3	2161
Buenos Aires	348.2	3854

# Using Spatial Operation in WHERE clause

**Query:** Find the names of all countries which are neighbors of the United States (USA) in the Country table.

```
SELECT C1.Name AS "Neighbors of USA"  
FROM Country C1,Country C2  
WHERE Touch(C1.Shape,C2.Shape)=1  
AND C2.Name ='USA '
```

Note: Spatial operator **Touch()** is used in **WHERE clause to join Country table with itself**. This query is an example of spatial self join operation.

# Spatial Query with multiple tables

**Query:** For all the rivers listed in the River table, find the countries through which they pass.

```
SELECT R.Name, C.Name  
FROM River R, Country C  
WHERE Cross(R.Shape,C.Shape)=1
```

Note: Spatial operation “Cross” is used to join River and Country tables. This query **represents a spatial join operation**.

## Example Spatial Query...Buffer and Overlap

**Query:** The St. Lawrence River can supply water to cities that are within 300 km. List the cities that can use water from the St. Lawrence.

```
SELECT Ci.Name  
FROM City Ci, River R  
WHERE Overlap(Ci.Shape, Buffer(R.Shape,300))=1  
AND R.Name = 'St.Lawrence'
```

# Recall List of Spatial Query Examples

- Simple SQL SELECT\_FROM\_WHERE examples
  - Spatial analysis operations
    - Unary operator: Area
    - Binary operator: Distance
  - Boolean Topological spatial operations - WHERE clause
    - Touch
    - Cross
  - Using spatial analysis and topological operations
    - Buffer, overlap
- Complex SQL examples
  - Aggregate SQL queries (Q9, pp. 70)
  - Nested queries (Q3 pp. 68, Q10, pp. 70)

# Using spatial operation in an aggregate query

**Query:** List all countries, ordered by number of neighboring countries.

```
SELECT Co.Name, Count(Co1.Name)  
FROM Country Co, Country Co1  
WHERE Touch(Co.Shape,Co1.Shape)  
GROUP BY Co.Name  
ORDER BY Count(Co1.Name)
```

Notes: This query can be used to differentiate querying capabilities of simple GIS software (e.g. ArcView) and a spatial database. It is quite tedious to carry out this query in GIS.

Earlier version of OGIS did not provide **spatial aggregate operation** to support GIS operations like reclassify.

# Using Spatial Operation in Nested Queries

**Query:** For each river, identify the closest city.

```
SELECT C1.Name, R1.Name  
FROM City C1, River R1  
WHERE Distance (C1.Shape,R1.Shape) <= ALL ( SELECT  
                                              Distance (C2 . Shape)  
FROM City C2  
WHERE C1.Name <> C2.Name  
 )
```

Note: Spatial operation Distance used in context of a nested query.

# Nested Spatial Query

**Query:** List the countries with only one neighboring country. A country is a neighbor of another country if their land masses share a boundary. According to this definition, island countries, like Iceland, have no neighbors.

```
SELECT Co.Name  
FROM Country Co  
WHERE Co.Name IN (SELECT Co.Name  
    FROM Country Co, Country Co1  
    WHERE Touch(Co.Shape, Co1.Shape)  
    GROUP BY Co.Name  
    HAVING Count(*)=1)
```

Note: It shows a complex nested query with aggregate operations. Such queries can be written into two expression, namely a view definition, and a query on the view. The inner query becomes a view and outer query is run on the view. This is illustrated in the next slide.

# Rewriting nested queries using Views

- Views are like tables
  - Represent derived data or result of a query
  - Can be used to simplify complex nested queries
  - Example follows:

**CREATE VIEW** Neighbor AS

**SELECT** Co.Name, Count(Co1.Name)AS num neighbors  
**FROM** Country Co,Country Co1  
**WHERE** Touch(Co.Shape,Co1.Shape)  
**GROUP BY** Co.Name

**SELECT** Co.Name,num neighbors

**FROM** Neighbor  
**WHERE** num neighbor = ( **SELECT** Max(num neighbors)  
                         **FROM** Neighbor )



Cairo University



# Geographic Information System And Spatial Databases

Course Code: IS443

## Chapter 4: Spatial Networks

# 5.1 Example Spatial Networks

## Road, River, Railway networks

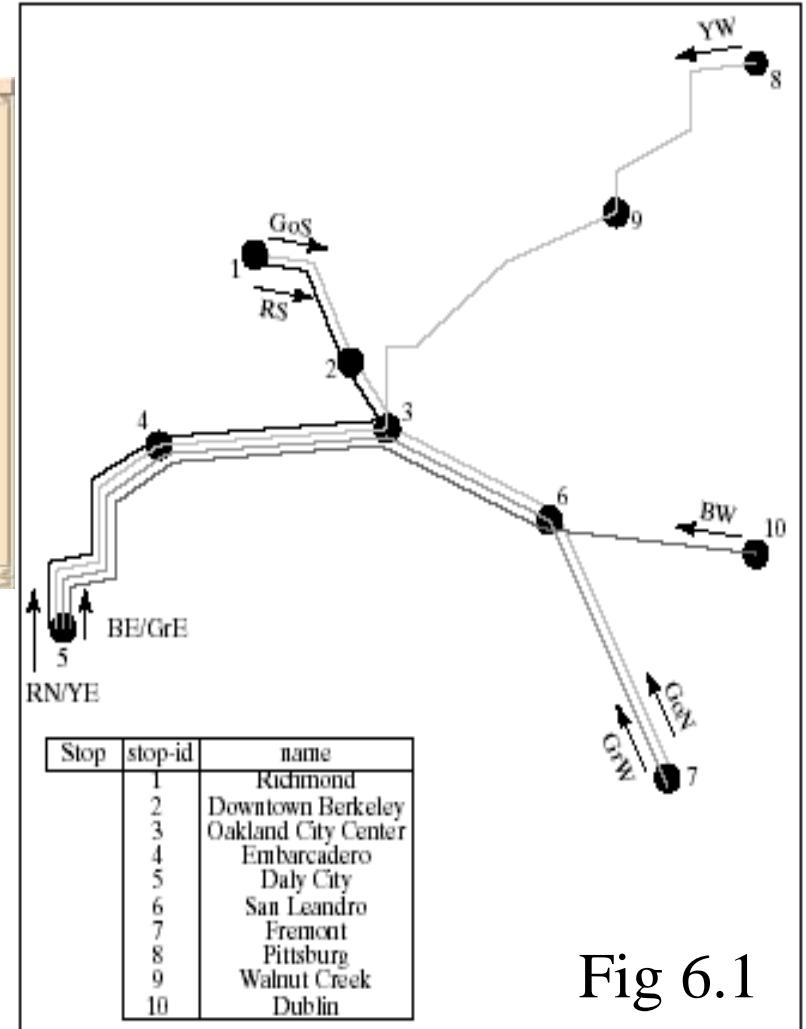
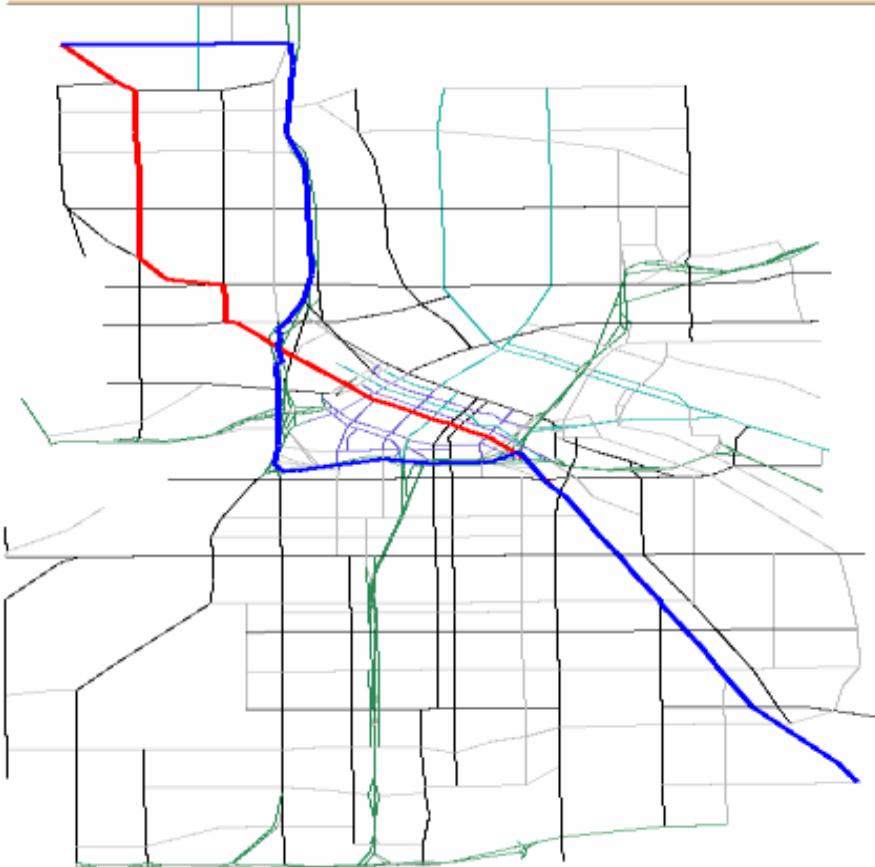


Fig 6.1

# Spatial network query example

File Display Options



## **Shortest path query**

Smallest travel time path between the two points shown in blue color. It follows a freeway (I-94) which is faster but not shorter in distance.

Shortest distance path between the same two points shown in red color, which is hidden under blue on common edges.

# Queries on Example Networks

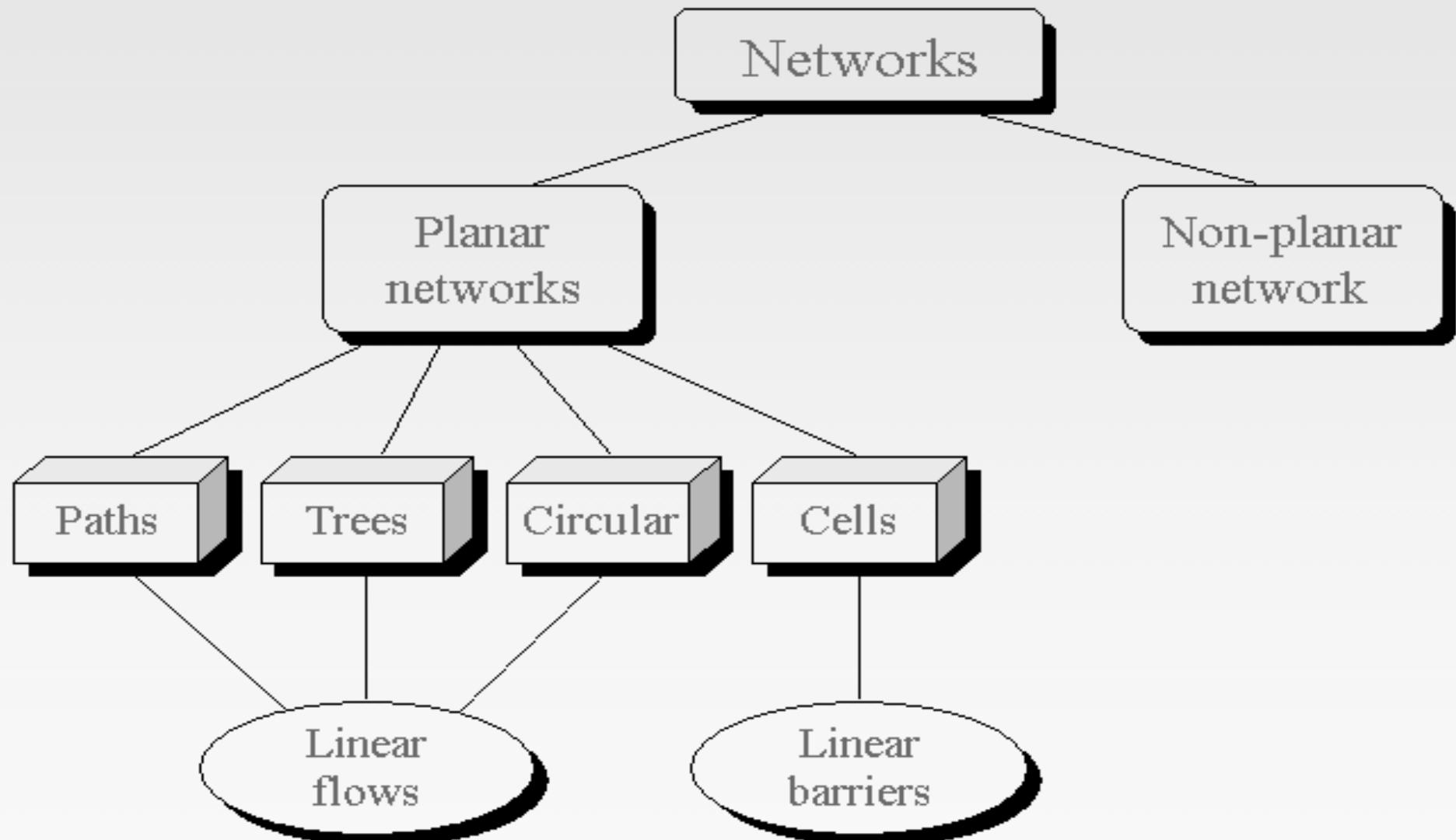
- Railway network
  - 1. Find the number of stops on the Yellow West (YW) route.
  - 2. List all stops which can be reached from Downtown Berkeley.
  - 3. List the route numbers that connect Downtown Berkeley and Daly City.
  - 4. Find the last stop on the Blue West (BW) route.
- River network
  - 1. List the names of all direct and indirect tributaries of the Mississippi river
  - 2. List the direct tributaries of the Colorado.
  - 3. Which rivers could be affected if there is a spill in river P1?
- Road Network
  - 1. Find shortest path from my current location to a destination.
  - 2. Find nearest hospital by distance along road networks.
  - 3. Find shortest route to deliver goods to a list of retail stores.
  - 4. Allocate customers to nearest service center using distance along roads

# Topological Classification of Networks

- ❑ Planar networks:  
No links intersect  
except at nodes
  - ❑ e.g. road and  
highway networks
  
- ❑ Non-planar  
networks: Links  
intersect
  - ❑ e.g. Airline  
networks



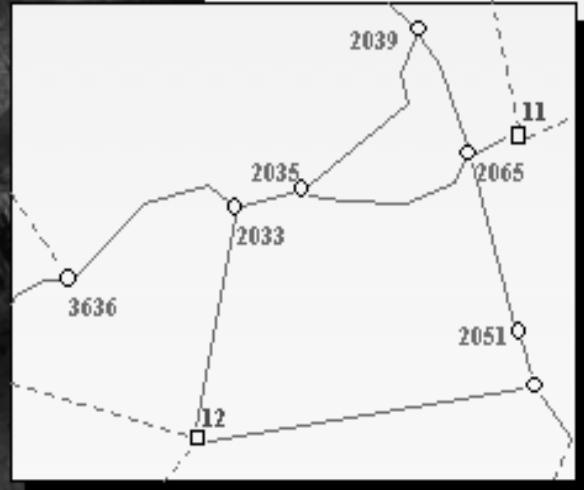
# Network Classification And Applications



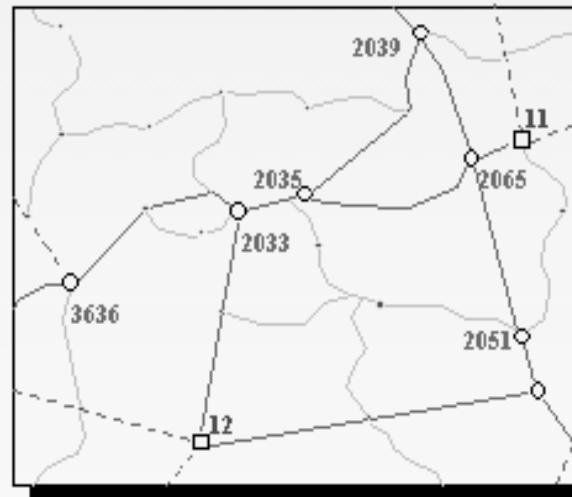
# Network Level of Details

- ❑ The fine level is nearly identical to the actual street network
- ❑ The medium level corresponds to transportation planning practice
- ❑ The coarse level only represents the arterial roads

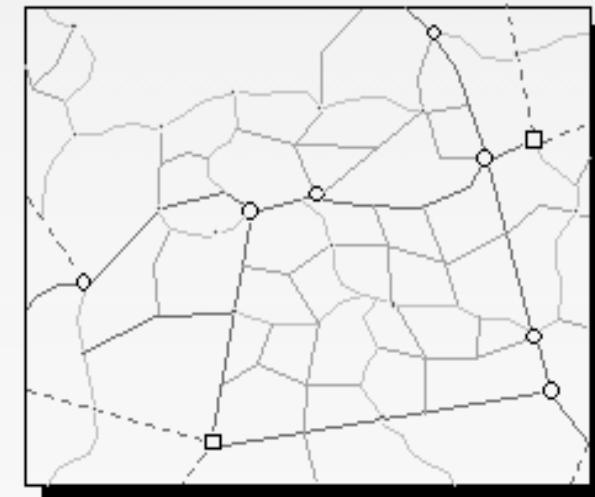
Coarse level network



Medium level network



Fine level network



# Link Versus Node Based Approaches

- ❑ Most networks are entered as line graphs in a vector GIS and attribute data is recorded with links rather than with nodes.
- ❑ Example for link description include traffic load, capacity of the road segment, etc.
- ❑ The link based approaches is not entirely suitable for applications that deal directly with nodes in a network.
- ❑ A typical example is the air passenger volume between cities where there is no direct link (multiple routes).

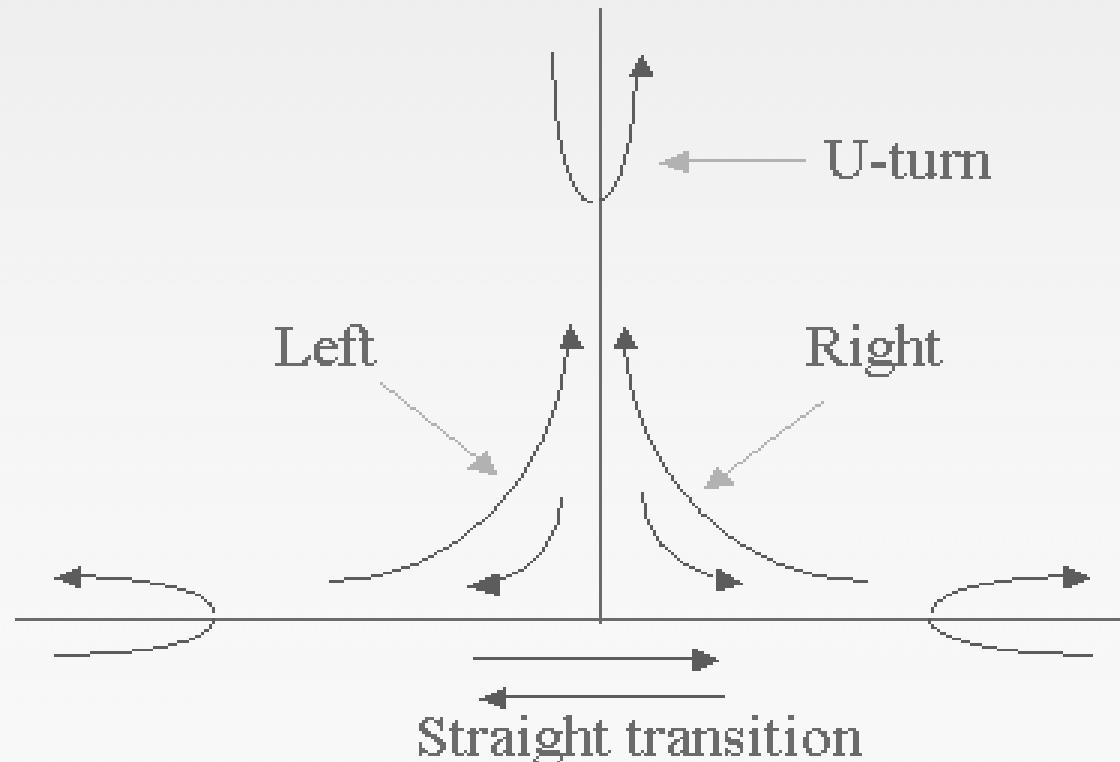
# Network Data Models

- ❑ Network Links
  - ❑ linear entities through which movement and communications takes place (e.g. road segment)
- ❑ Network Nodes
  - ❑ end points of network links (e.g. towns, cities, etc)
- ❑ Stops
  - ❑ locations visited in a path (e.g. customers on a delivery route)
- ❑ Centres
  - ❑ discrete locations where there exists a supply or attraction (e.g. shopping centres, airports, fire stations)

# Turns

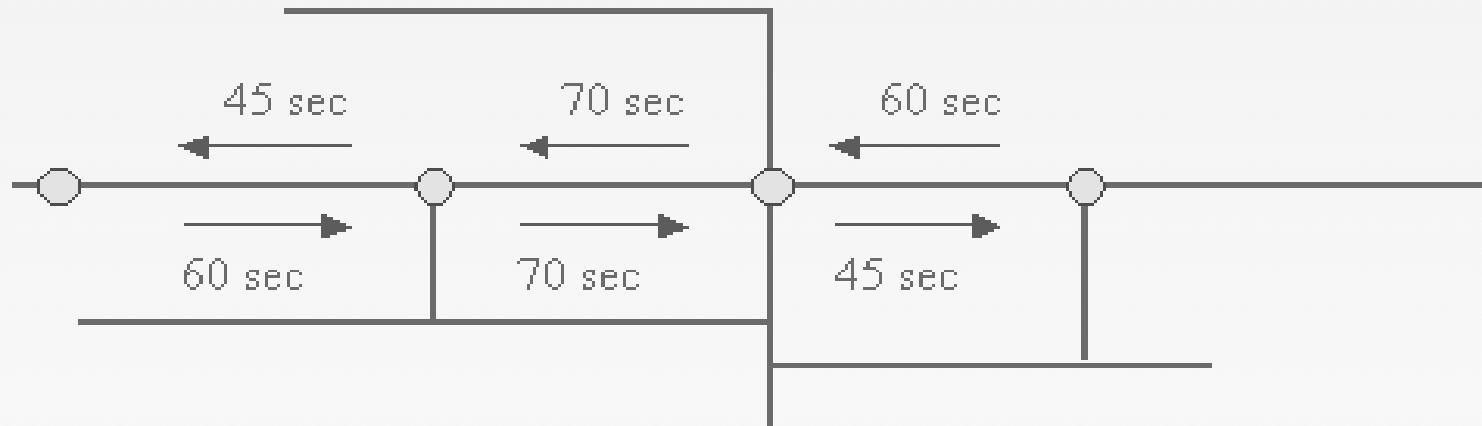
- ❑ represent relationships between network links
- ❑ can affect movement through a network (e.g. a right hand turn against on-coming traffic takes more time than proceeding straight through)

There are  $n^2$  possible turns at every network node, where  $n$  is the number of links connected at that node. Even at a node with a single link, it is possible to make one u-turn.



# Links

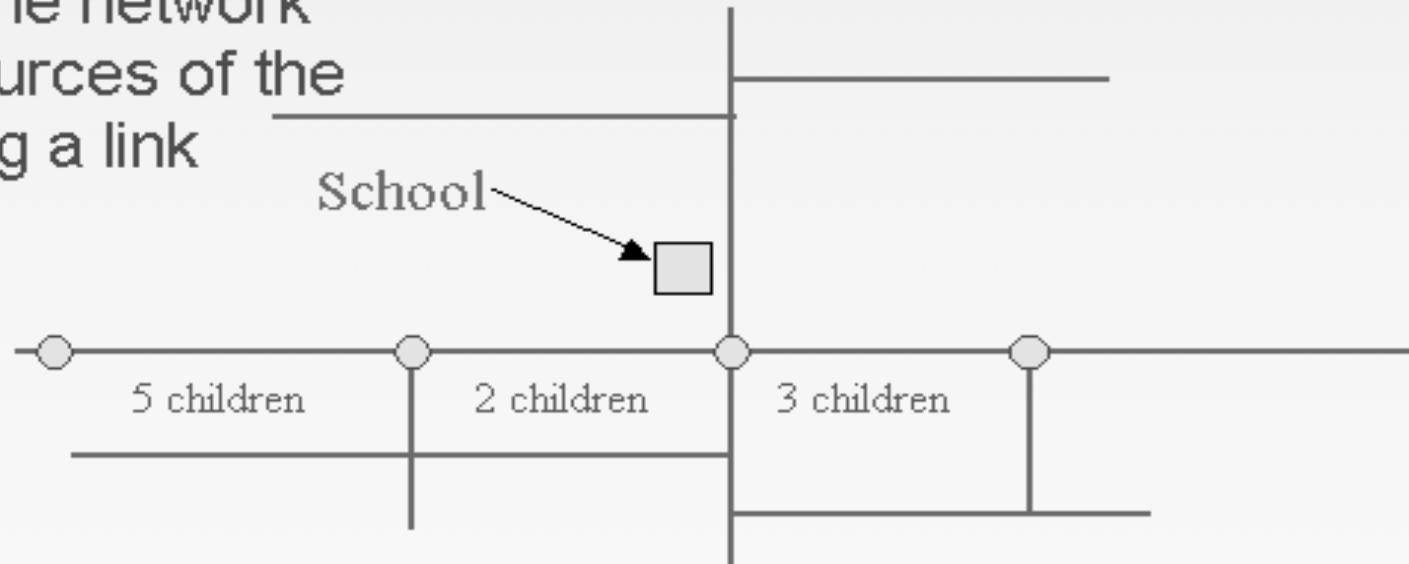
- ❑ Link Attributes
  - ❑ attributes describing links (e.g. one- or two-way, number of lanes, etc.)
- ❑ Link Impedance
  - ❑ the cost of moving on a network link, usually given in terms of time or money cost or a generalised cost
  - ❑ The impedance depends on the direction of travel



# Link Demand

- ❑ the level of the resource that is associated with a particular network link.
  - ❑ e.g. the number of school children residing on a street (link demand) which are assigned to a centre (school).

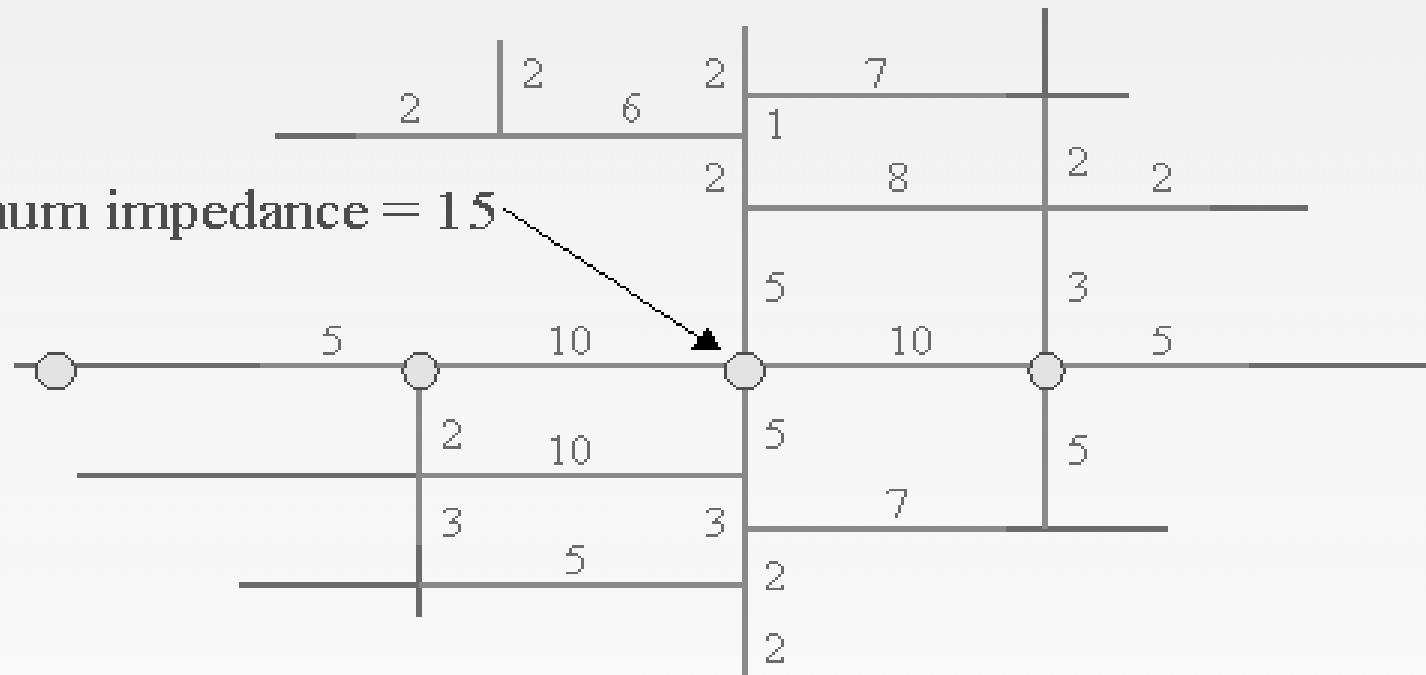
The children who live along the streets of the network utilise the resources of the school, creating a link demand



# Centres

- ❑ Supply: total quantity of resource available to satisfy the demand associated with links of network
  - ❑ Maximum Impedance: is the maximum total impedance that may be encountered between the centre and the end of any allocated sequence of links

Maximum impedance = 15



# Analysis Capabilities

- ❑ Location/Allocation
  - ❑ optimally locating a set of objects so that variable or variables will achieve a maximum or minimum value
- ❑ Routing
  - ❑ finding shortest (or cheapest or quickest) route between locations
- ❑ Accessibility
  - ❑ providing an aggregate measure of how accessible a location is to other locations
- ❑ Address Matching
  - ❑ finding spatial locations based on address descriptions

# Location/allocation

## □ The problem to be addressed

### □ Private sector

- e.g. optimum location of a factory is required that will minimise the cost of getting products to consumers
- e.g. locating branch plants so that total transport cost to the market from both plants is a minimum
- e.g. location of two new warehouses in a certain territory in order to minimise transportation costs

### □ Public sector

- Ordinary services: health, education and welfare facilities
- Emergency services: medical, ambulance, police, fire, vehicle breakdown services

### □ Facilities should be located so as to be the most accessible

# Meaning of Most Accessible

- A location pattern is most acceptable to people when:
  - the total of the distances of all people from their closest facility is minimum (aggregate distance minimisation criterion).
  - the farthest distance of people from their closest facility is minimum (minimax distance criterion).
  - the number of people in the proximal area surrounding each facility is approximately equal (equal assignment criterion).
  - the number of people in the proximal area surrounding each facility is always greater than a specified number (threshold constraint criterion).
  - the number of people in the proximal area surrounding each facility is never greater than a specified number (capacity constraint criterion).

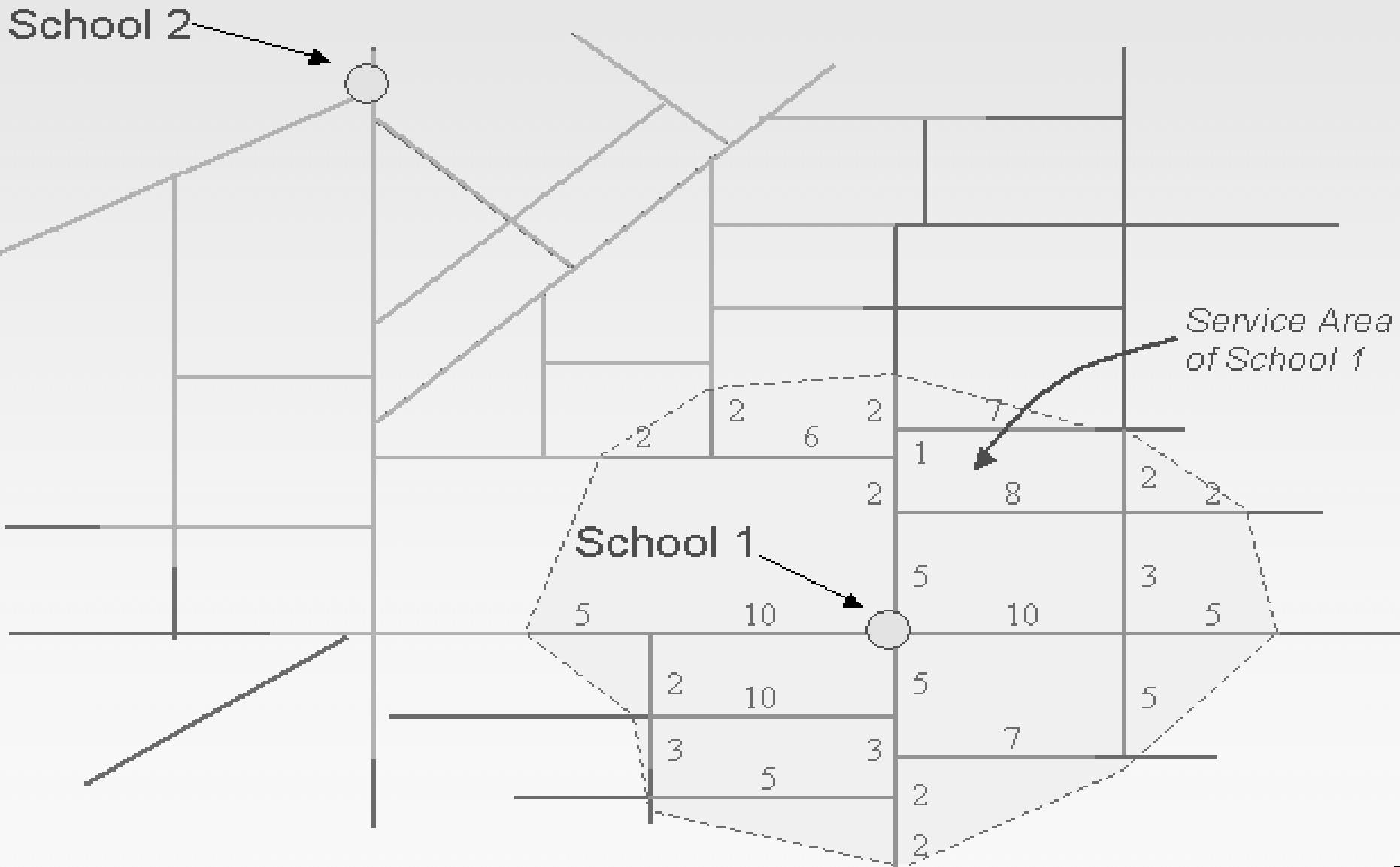
# Location

- ❑ Single source location problems
  - Given distances defined between each node point of a route network, find a location from which the sum of the distances to all other points is least (median of the graph).
- ❑ Multi-facility location
  - Minimising average distance
  - Minimising the maximum distance to closest supply centre
  - Minimising the number of centres required for every demand point to be within a critical distance of a supply point
  - Minimising average distance subject to a maximum distance constraint

# Allocation

- ❑ Which centre serves a particular link?
- ❑ Two criteria are used for allocation:
  - ❑ supply: amount of resource a facility (centre) has available
  - ❑ demand: amount of the resource demanded on the links or nodes
- ❑ Allocation works by assigning demand, usually to the nearest centre, until the demand matches the supply of the centre.
  - ❑ e.g. A school (centre) has a quantity of spaces available for children. The number of children living on each street form the demand. Streets are assigned to the school until the total number of children on those streets equals the number of available spaces.

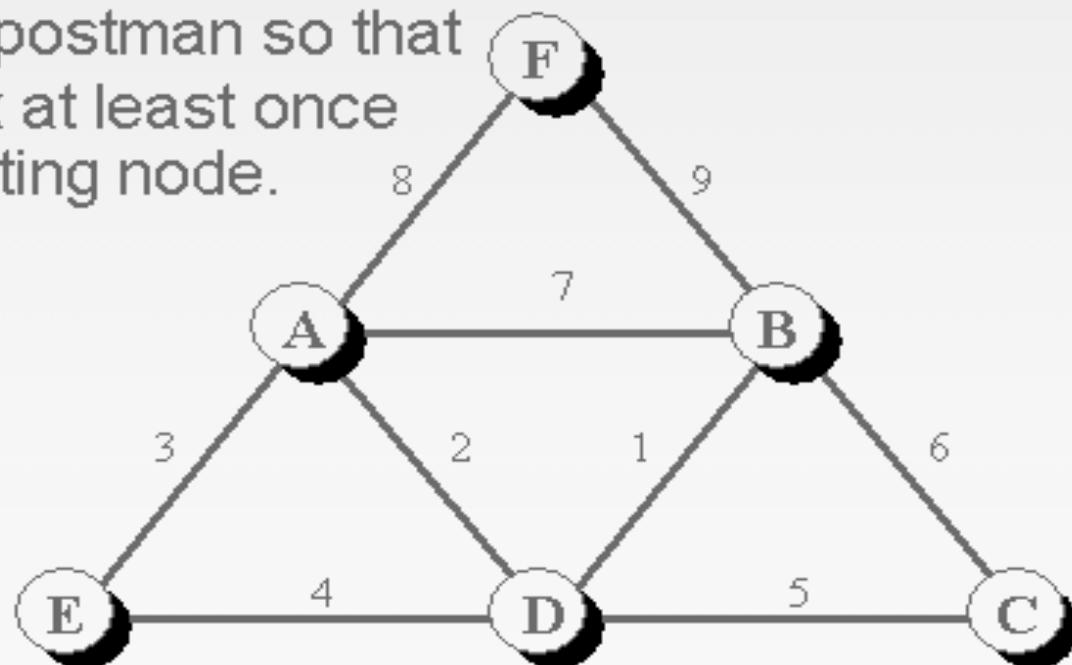
# Service Area



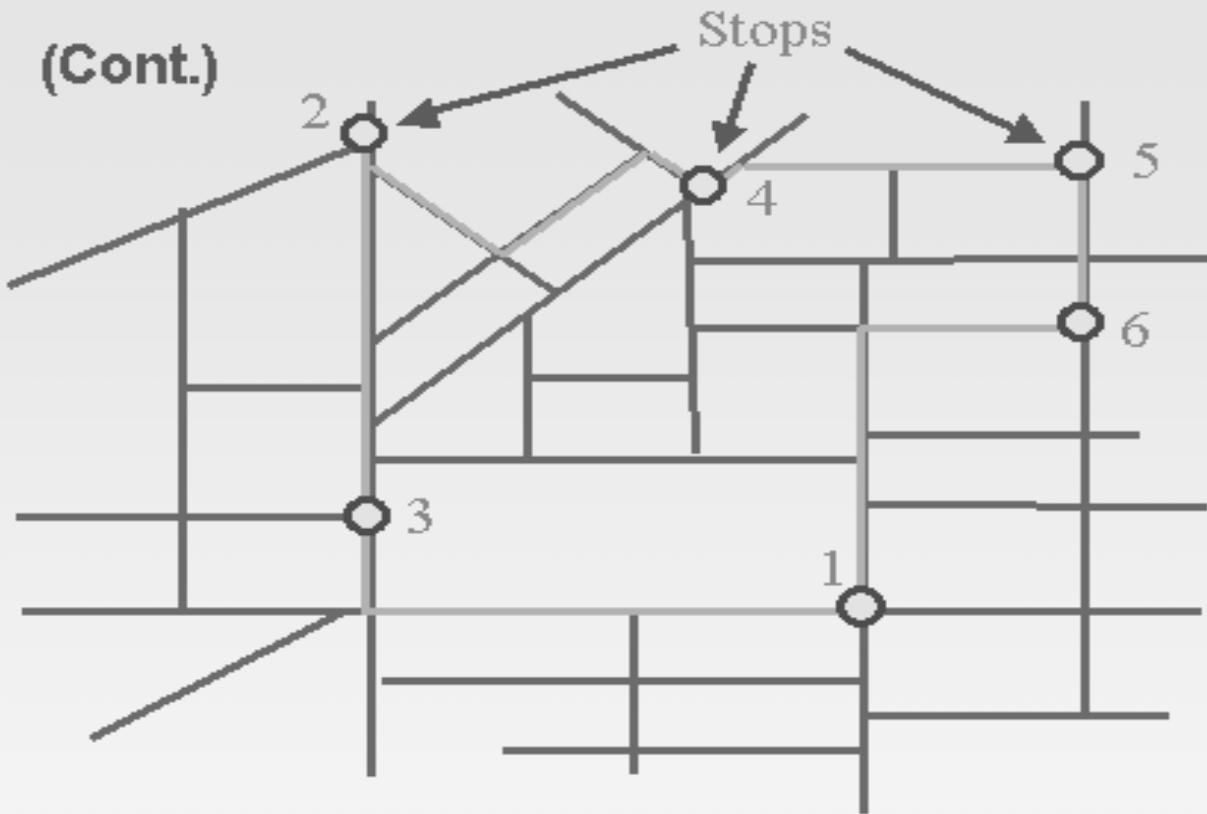
# Routing

Postman problem:

- how to cover all the streets in the route and return back to the starting point with as little travelling as possible.
- This is the problem of finding the shortest route for the postman so that he traverses each link at least once and returns to his starting node.



# Routing (Cont.)



Travelling salesman problem:

- to call at each town before returning home.
- This is the problem of finding a route that minimises the total distance (or time or cost) needed to visit all the towns in his district.

# Accessibility

- Accessibility provides an aggregate measure of how accessible a location is to other locations. It can be defined as the ease of participating in activities.
- Types of accessibility measures:
  - Topological accessibility: states whether two points in space are physically connected by a transport system thus enabling movement to take place between them.
  - Relative accessibility: a measure of the degree of connectivity or accessibility between places.
  - Integral accessibility: measures the accessibility of a site to a number of other sites or activities.

# Accessibility (Cont.)

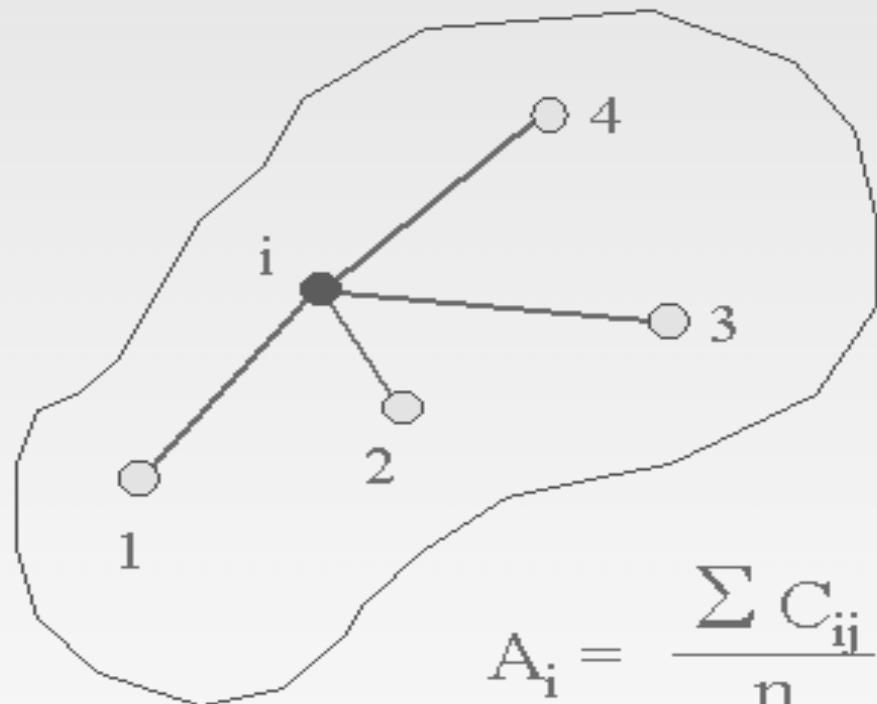
Relative Accessibility



$$A_i = C_{ij}$$

e.g. travel time to nearest health clinic  
distance to Central Business District

Integral Accessibility



$$A_i = \frac{\sum C_{ij}}{n}$$

e.g. mean travel time to all health clinics in the region  
mean distance to all other zones

# Computing Origin Accessibility

- The basic principle is that the effect of one location on another is directly proportional to its supply (attractiveness) and inversely proportional to its distance

		Destination	
		M1	M2
Origin	A	0.5	0.7
	B	1.6	2.1

The distance between the origins A and B and destinations M1 and M2 in kilometres

Attraction	
M1	M2
3.0	5.0

Production	
A	B
2	3

To compute accessibility, each market is given an index of how attractive it is and the population (or production) at each location is determined

# Computing Origin Accessibility

- Assume a distance decay exponent of 2, the accessibility of A to M1 is computed as:

attractiveness index of market centre

(distance between location and market)<sup>2</sup>

- The total accessibility can be expressed as:

where

$$P_i = \sum_{j=1}^n \frac{W_j}{d_{ij}^\beta}$$

is the accessibility at point  $i$   
is the attractiveness of location  $j$   
is the distance between location  $i$  and  $j$   
is the exponent for distance decay  
is the number of locations in the region

# Spatial Interaction

- ❑ How accessible is one location to all other locations?
- ❑ Computing interaction

$$\text{Interaction} = \frac{\text{production} \times \text{attractiveness}}{(\text{distance between origin and destination})^2}$$

		Destination	
		M1	M2
Origin	A	24.0	20.4
	B	3.5	3.4

The raw interactions between the two locations A and B and centres M1 and M2 result in a 2 by 2 matrix.

# 5.2 Spatial Network Data Models

- Recall 3 level Database Design
  - Conceptual Data Model
    - Graphs
  - Logical Data Model -
    - Data types - Graph, Vertex, Edge, Path, ...
    - Operations - Connected(), Shortest\_Path(), ...
  - Physical Data Model
    - Record and file representation - adjacency list
    - File-structures and access methods - CCAM

# 5.2 Conceptual Data Models

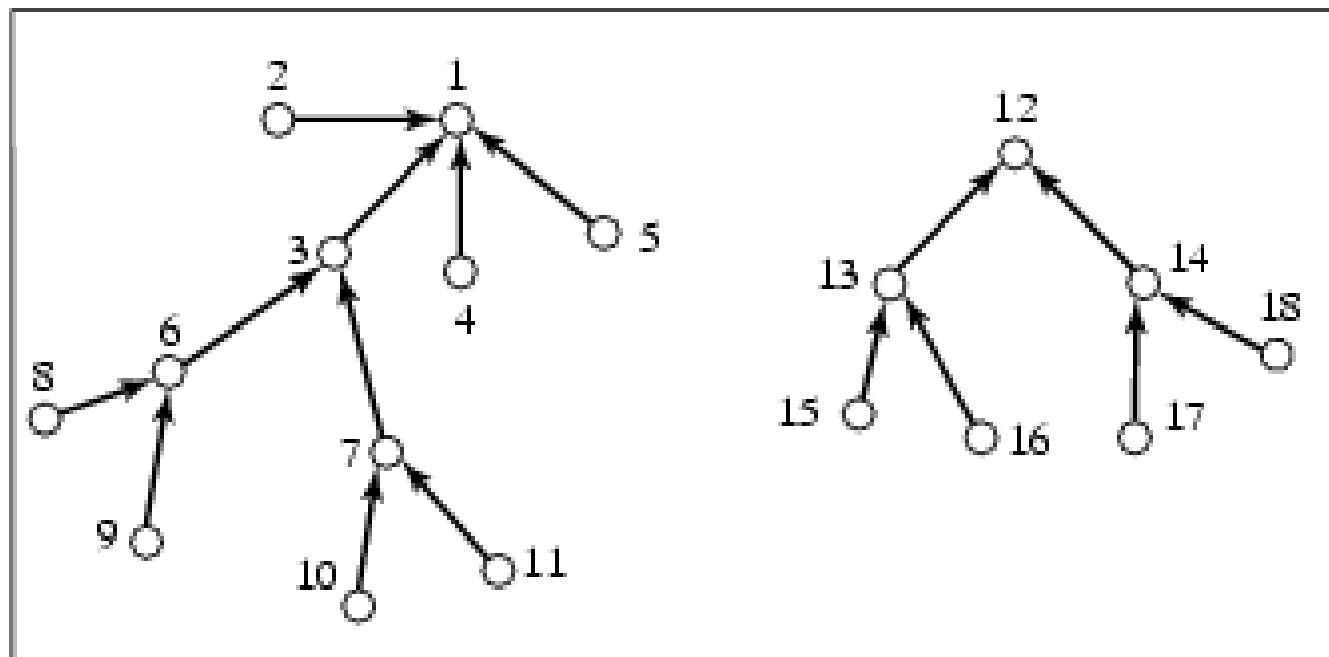
- Conceptual Data Model for Spatial Networks
  - A *graph*,  $G = (V, E)$
  - $V$  = a finite set of *vertices*
  - $E$  = a set of edges  $E$ , between vertices in  $V$
- Example: two graph models of a roadmap
  - 1. Nodes = road-intersections, edges = road segment between intersections
  - 2. Nodes = roads (e.g. Route 66), edge(A, B) = road A crosses road B
- Classifying graph models
  - Do nodes represent spatial points? - spatial vs. abstract graphs
  - Are vertex-pair in an edge order? - directed vs. undirected
- Example (continued)
  - Model 1 is a spatial graph, Model 2 is an abstract graph
  - Model 2 is undirected but can be directed or undirected

## 5.2 Conceptual Data Model - Exercise

• Exercise: Review the graph model of river network in Figure 6.3 (pp. 157).

- List the nodes and edges in the graph.
- List 2 paths in this graph.
- Is it spatial graph? Justify your answer.
- Is it a Directed graph? Justify your answer.

Fig 6.3



# 5.2 Logical Data Model - Data types

- Common data types
  - Vertex: attributes are label, isVisited, (location for spatial graphs)
  - DirectedEdge : attributes are start node, end node, label
  - Graph : attributes are setOfVertices, setOfDirectedEdges, ...
  - Path : attributes are sequenceOfVertices
- Questions. Use above data types to model.
  - an undirected edge
  - train routes in BART network (Fig. 6.1(a), pp. 150)
  - rivers in the river network (Fig. 6.1(b), pp.150)
  - Note: Multiple distinct solutions are possible in last two cases!

# 5.2 Logical Data Model - Operations

- Low level operations on a Graph G

- IsDirected - return true if and only if G is directed
- Add , AddEdge - adds a given vertex, edge to G
- Delete, DeleteEdge - removes specifies node (and related edges), edge from G
- Get, GetEdge - return label of given vertex, edge
- Get-a-successor, GetPredecessors - return start or end vertex of an edge
- GetSuccessors - return end vertices of all edge starting at a given vertex

- Building blocks for queries

- shortest\_path(vertex start, vertex end)
- shortest tour(vertex start, vertex end, setOfVertices stops)
- locate\_nearest\_server(vertex client, setOfVertices servers)
- allocate(setOfVertices servers)

```
public class Graph
{
    public void add(Object label);
    // label represents the vertex to be added

    public void addEdge(Object v1, Object v2, Object label);
    // an edge is inserted between the two vertices v1 and v2

    public Object delete(Object label);
    // removes a vertex

    public Object deleteEdge(Object v1, Object v2);
    // the edge spanned by vertices v1 and v2 is removed

    public Object get(Object label);
    // the label of the vertex is returned

    public Edge getEdge(Object v1, Object v2);
    // the edge spanned by vertices v1 and v2 is returned

    public Object get-a-Successor(Object label);
    // an adjacent node of the vertex is returned
```

```
public Iterator getSuccessors(Object label);
// all the adjacent neighbors are returned

public Iterator getPredecessors(Object label);
// all the parents are returned

public boolean isDirected();
// returns true if the graph is directed

}
```

```
public class Vertex
{
    public Vertex(Object label)
        // the constructor for the class. A node with the appropriate
        label is created.

    public Object label()
        // returns a label associated with the vertex

    public boolean visit()
        // marks the vertex as being visited.

    public boolean isVisited()
        // returns true if and only if the vertex has been visited.

}
```

```
public class Edge
{
    public Edge(Object v1, Object v2, Object label, boolean directed)
        // the constructor for the class. The edge is "directed"
        // if set true.

    public Object start()
        // returns the first node of the edge

    public Object end()
        // returns the second node of the edge
}
```

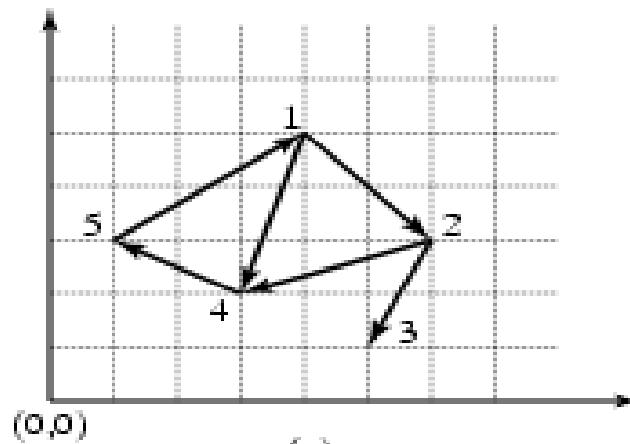
# 5.2 Physical Data Models

- Categories of record/file representations
  - Main memory based
  - Disk based
- Main memory representations of graphs
  - Adjacency matrix  $M[A, B] = 1$  if and only if edge(vertex A, vertex B) exists
  - Adjacency list : maps vertex A to a list of successors of A
  - Example: See Figure 6.2(a), (b) and (c) on next slide
- Disk based
  - normalized - tables, one for vertices, other for edges
  - denormalized - one table for nodes with adjacency lists
  - Example: See Figure 6.2(a), (d) and (e) on next slide

# 5.2.2 Physical Data Models -

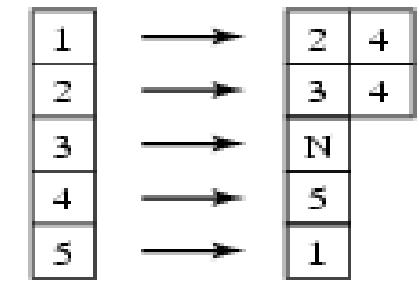
## Figure 6.2

Fig 6.2



		Destination				
		1	2	3	4	5
source	1	0	1	0	1	0
	2	0	0	1	1	0
	3	0	0	0	0	0
	4	0	0	0	0	1
	5	1	0	0	0	0

(b) Adjacency-matrix



(c) Adjacency-List

Node (R)		
id	x	y
1	4.0	5.0
2	6.0	3.0
3	5.0	1.0
4	3.0	2.0
5	1.0	3.0

Edge (S)		
source	dest	distance
1	2	$\sqrt{8}$
1	4	$\sqrt{10}$
2	3	$\sqrt{5}$
2	4	$\sqrt{10}$
4	5	$\sqrt{5}$
5	1	$\sqrt{18}$

(d) Node and Edge Relations

id	x	y	Successors	Predecessors
1	4.0	5.0	(2,4)	(5)
2	6.0	3.0	(3,4)	(1)
3	5.0	1.0	()	(2)
4	3.0	2.0	(5)	(1,2)
5	1.0	3.0	(1)	(4)

(e) Denormalized Node Table

# 5.2 Case Studies Revisited

- Goal: Compare relational schemas for spatial networks
  - River networks has an edge table, FallsInto
  - BART train network does not have an edge table
  - Edge table is crucial for using SQL transitive closure .
- Representation of river networks
  - Conceptual : abstract graph
    - nodes = rivers
    - directedEdges(R1, R2) if and only R1 falls into R2
  - Table representation given in Table
- Representation of BART train network
  - Conceptual :
    - entities = Stop, Directed Route
    - many to many relationship: aMemberOf( Stop, Route)

TABLE 6.2: The RouteStop Table of BART

RouteStop	routenumber	stopid	rank
	1	1	1
	1	2	2
	1	3	3
	1	4	4
	1	5	5
2	5	1	
2	4	2	
2	3	3	
2	2	4	
2	1	5	
3	1	1	
3	2	2	
3	3	3	
3	6	4	
3	7	5	
4	7	1	
4	6	2	
4	3	3	
4	2	4	
4	1	5	
5	8	1	
5	9	2	
5	3	3	
5	4	4	
5	5	5	
6	5	1	
6	4	2	
6	3	3	
6	9	4	
6	8	5	
7	10	1	
7	6	2	
7	3	3	
7	4	4	
7	5	5	
8	5	1	
8	4	2	
8	3	3	
8	6	4	
8	10	5	
9	7	1	
9	6	2	
9	4	3	
9	5	4	
10	5	1	
10	6	2	
10	10	3	
10	7	4	

TABLE 6.1: The Stop and DirectedRoute Tables of BART

Stop	stopid	name
1	Richmond	
2	Downtown Berkeley	
3	Oakland City Center	
4	Embarcadero	
5	Daly City	
6	San Leandro	
7	Fremont	
8	Pittsburg	
9	Walnut Creek	
10	Dublin	

DirectedRoute	number	name
1	Red South	
2	Red North	
3	Gold South	
4	Gold North	
5	Yellow West	
6	Yellow East	
7	Blue West	
8	Blue East	
9	Green West	
10	Green East	

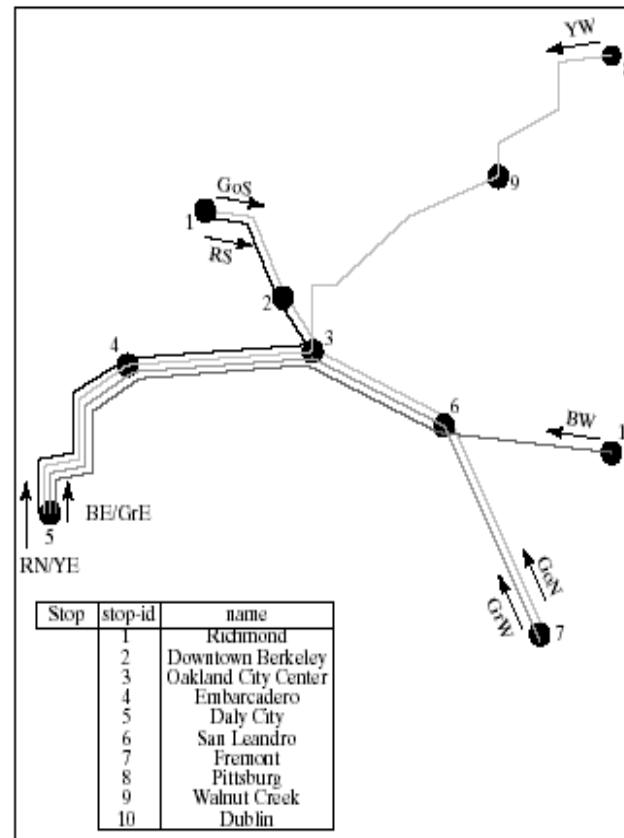


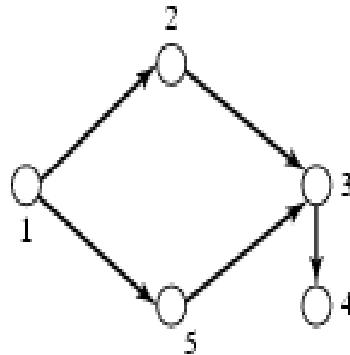
TABLE 6.3: The River and FallsInto Relations of River Network



River	riverid	name	FallsInto	source	dest
	1	Mississippi		2	1
	2	Ohio		3	1
	3	Missouri		4	1
	4	Red		5	1
	5	Arkansas		6	3
	6	Platte		7	3
	7	Yellowstone		8	6
	8	P1		9	6
	9	P2		10	7
	10	Y1		11	7
	11	Y2		13	12
	12	Colorado		14	12
	13	Green		15	13
	14	Gila		16	13
	15	G1		17	14
	16	G2		18	14
	17	G11			
	18	G12			

# Concept of Transitive Closure

- Consider a graph  $G = (V, E)$
- Let  $G^*$  = Transitive closure of  $G$
- Then  $T = \text{graph } (V^*, E^*)$ , where
  - $V^* = V$
  - $(A, B) \in E^*$  if and only if there is a path from A to B in G.

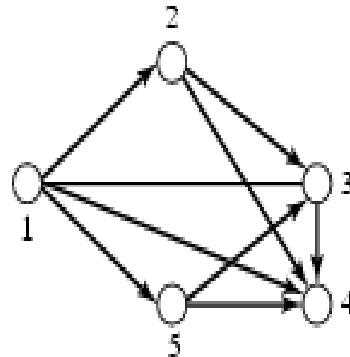


(a) Graph G

R	
SOURCE	DEST
1	2
1	5
2	3
3	4
5	4

(b) Relation form

- Example G has 5 nodes and 5 edges
  - $G^*$  has 5 nodes and 9 edges
  - Note edge (1,4) in  $G^*$  for path (1, 2, 3, 4) in G.



(c) Transitive closure ( $G$ ) = Graph  $G^*$

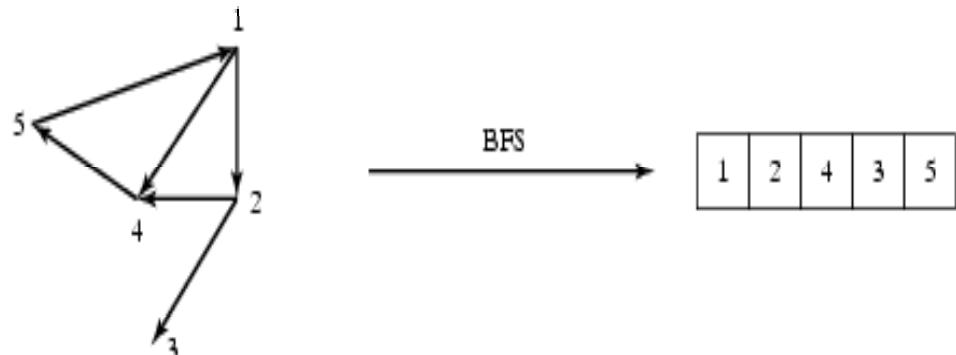
X	
SOURCE	DEST
1	2
1	5
2	3
3	4
5	3
1	3
2	4
5	4
1	4

(d) Transitive closure in relational form

## 5.4.2 Strategies for Connectivity Query

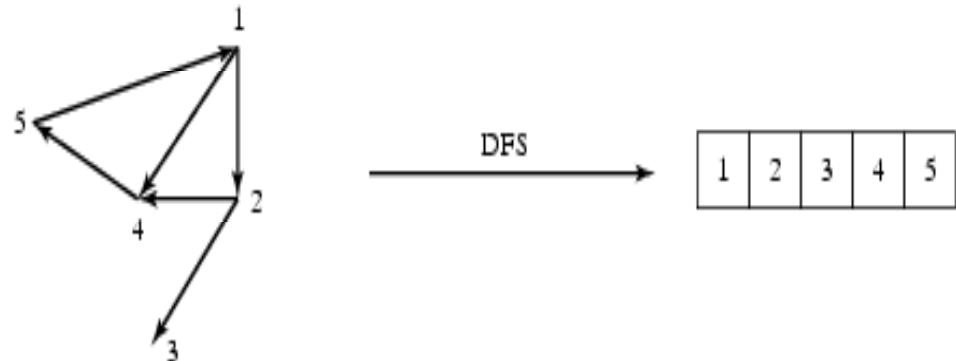
- Breadth first search -

- Visit descendent by generation
- children before grandchildren
- Example: 1 - (2,4) - (3, 5)



- Depth first search - basic idea

- Try a path till deadend
- Backtrack to try different paths
- Like a maze game
- Example: 1-2-3-2-4-5
- Note backtrack from 3 to 2

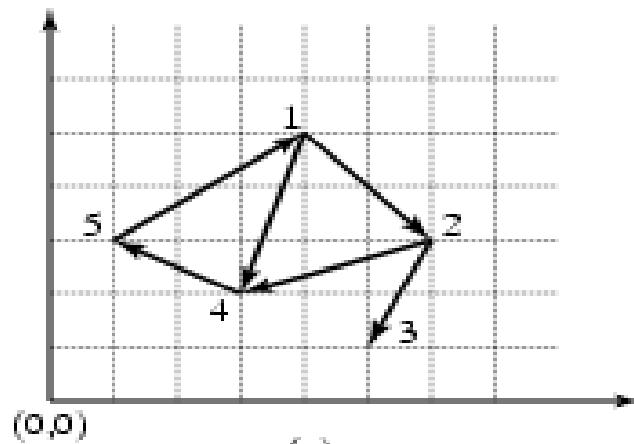


## 5.4.2 Shortest Path strategies -1

- Dijktra's algorithm
  - Identify paths to descendent by depth first search
  - Each iteration
    - Expand descendent with smallest cost path so far
    - Update current best path to each node, if a better path is found
  - Till destination node is expanded
- Proof of correctness is based on assumption of positive edge costs
- Example:
  - Consider shortest\_path(1,5) for graph in Figure 6.2(a), pp. 154
  - Iteration 1 expands node 1 and edges (1,2), (1,4)
    - set  $\text{cost}(1,2) = \sqrt{8}$ ;  $\text{cost}(1,4) = \sqrt{10}$  using Edge table in Fig. 6.2(d)
  - Iteration 2 expands least cost node 2 and edges (2,3), (2,4)
    - set  $\text{cost}(1,3) = \sqrt{8} + \sqrt{5}$
  - Iteration 3 expands least cost node 4 and edges (4,5)
    - set  $\text{cost}(1,5) = \sqrt{10} + \sqrt{5}$
  - Iteration 4 expands node 3 and Iteration 5 stops node 5.
  - Answer is the path (1-4-5)

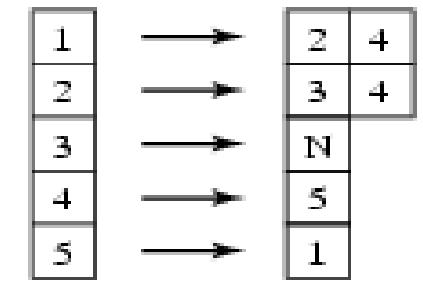
# Figure 6.2 for examples

Fig 6.2



		Destination				
		1	2	3	4	5
source	1	0	1	0	1	0
	2	0	0	1	1	0
	3	0	0	0	0	0
	4	0	0	0	0	1
	5	1	0	0	0	0

(b) Adjacency-matrix



(c) Adjacency-List

Node (R)		
id	x	y
1	4.0	5.0
2	6.0	3.0
3	5.0	1.0
4	3.0	2.0
5	1.0	3.0

Edge (S)		
source	dest	distance
1	2	$\sqrt{8}$
1	4	$\sqrt{10}$
2	3	$\sqrt{5}$
2	4	$\sqrt{10}$
4	5	$\sqrt{5}$
5	1	$\sqrt{18}$

(d) Node and Edge Relations

id	x	y	Successors	Predecessors
1	4.0	5.0	(2,4)	(5)
2	6.0	3.0	(3,4)	(1)
3	5.0	1.0	()	(2)
4	3.0	2.0	(5)	(1,2)
5	1.0	3.0	(1)	(4)

(e) Denormalized Node Table

## 5.4.2 Shortest Path Strategies-2

- Best first algorithm
  - Similar to Dijkstra's algorithm with one change
  - $\text{Cost}(\text{node}) = \text{actual\_cost}(\text{source}, \text{node}) + \text{estimated\_cost}(\text{node}, \text{destination})$
  - $\text{estimated\_cost}$  should be an underestimate of actual cost
    - Example - euclidean distance
- Given effective  $\text{estimated\_cost}()$  function, it is faster than Dijkstra's algorithm
- Example:
  - Revisit  $\text{shortest\_path}(1,5)$  for graph in Figure 6.2(a), pp. 154
  - Iteration 1 expands node 1 and edges (1,2), (1,4)
    - set  $\text{actual\_cost}(1,2) = \sqrt{8}$ ;  $\text{actual\_cost}(1,4) = \sqrt{10}$ ;
    - $\text{estimated\_cost}(2,5) = 5$ ;  $\text{estimated\_cost}(4,5) = \sqrt{5}$
  - Iteration 2 expands least cost node 4 and edges (4,5)
    - set  $\text{actual\_cost}(1,5) = \sqrt{10} + \sqrt{5}$ ,  $\text{estimated\_cost}(5,5) = 0$
  - Iteration 3 expands node 5
  - Answer is the path (1-4-5)

## 5.4.2 Shortest Path Strategies-3

- Dijktra's and Best first algorithms
  - Work well when entire graph is loaded in main memory
  - Otherwise their performance degrades substantially
- Hierarchical Routing Algorithms
  - Works with graphs on secondary storage
  - Loads small pieces of the graph in main memories
  - Can compute least cost routes
- Key ideas behind Hierarchical Routing Algorithm
  - Fragment graphs - pieces of original graph obtained via node partitioning
  - Boundary nodes - nodes of with edges to two fragments
  - Boundary graph - a summary of original graph
    - Contains Boundary nodes
    - Boundary edges: edges across fragments or paths within a fragment

## 6.4.2 Shortest Path Strategies-3

- Insight:

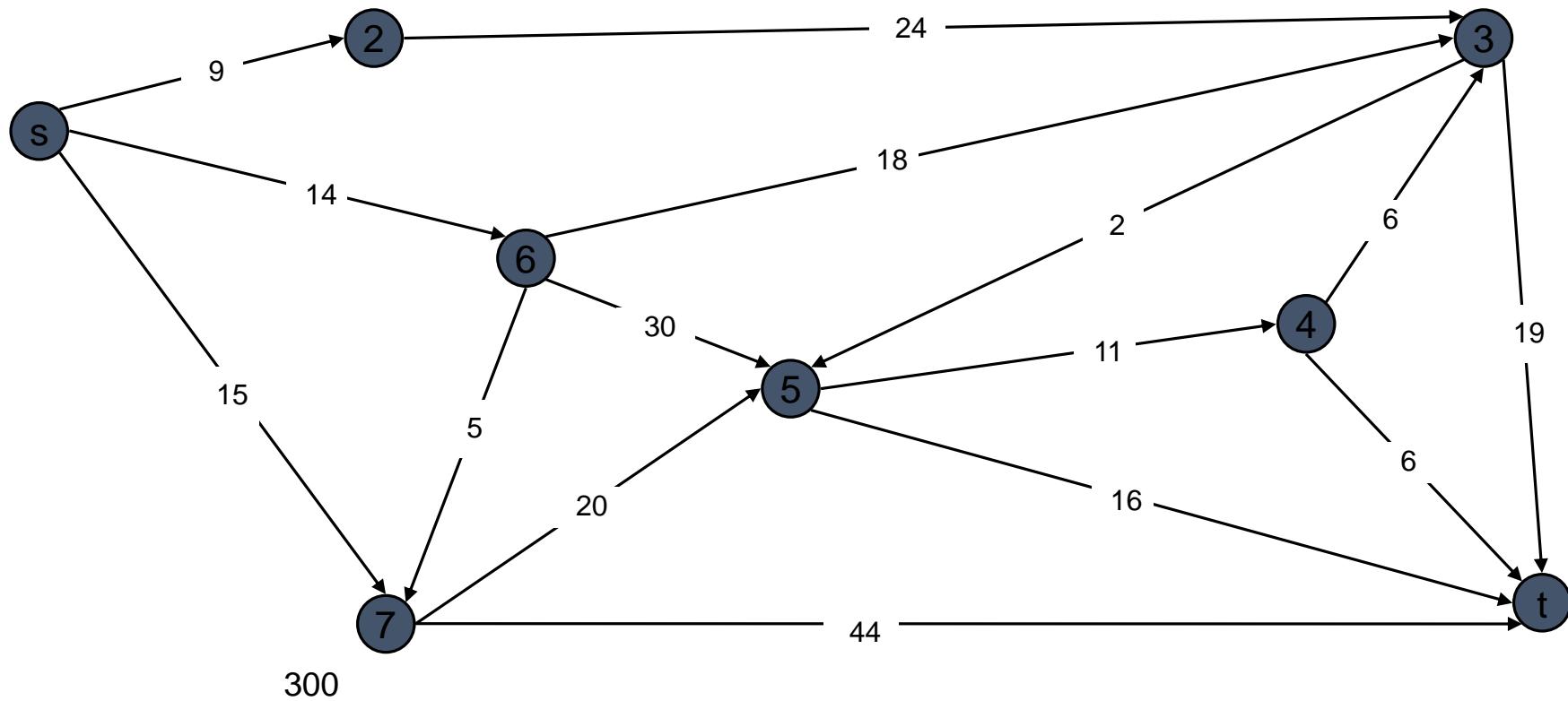
- A Summary of Optimal path in original graph can be computed
  - using Boundary graph and 2 fragments
- The summary can be expanded into optimal path in original graph
  - examining fragments overlapping with the path
  - loading one fragment in memory at a time
- See Theorems 1 and 2 (pp.170-171, section 6.4.4)

- Illustration of the algorithm

- Figure 6.7(a) - fragments of source and destination nodes
- Figure 6.7(b) - computing summary of optimal path using
  - Boundary graph and 2 fragments
  - Note use of boundary edges only in the path computation
- Figure 6.8(a) - The summary of optimal path using boundary edges
- Figure 6.8(b) Expansion back to optimal path in original graph

# Dijkstra's Shortest Path Algorithm (single source queries)

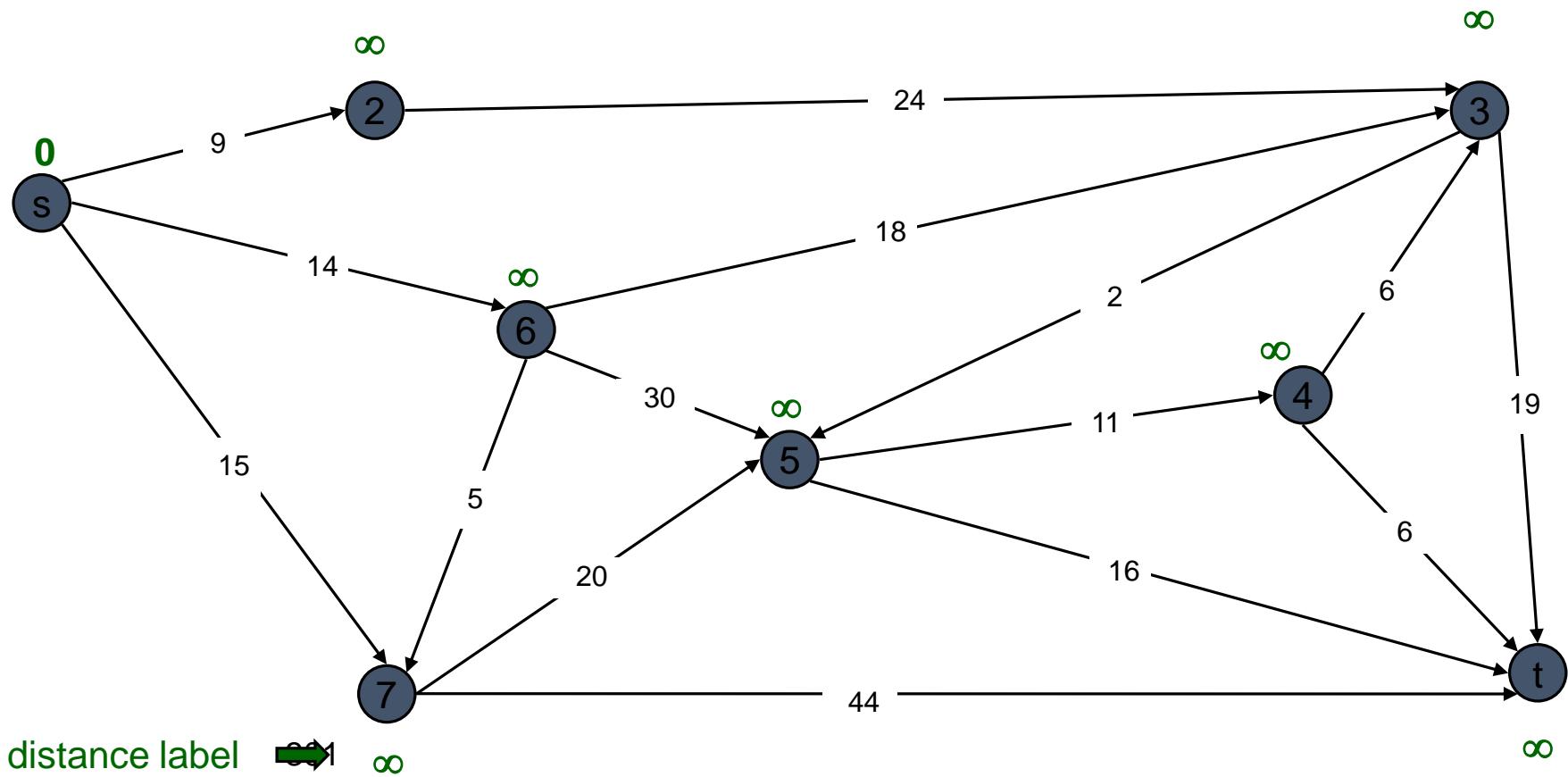
- Find shortest path from s to t.



# Dijkstra's Shortest Path Algorithm

S = { }

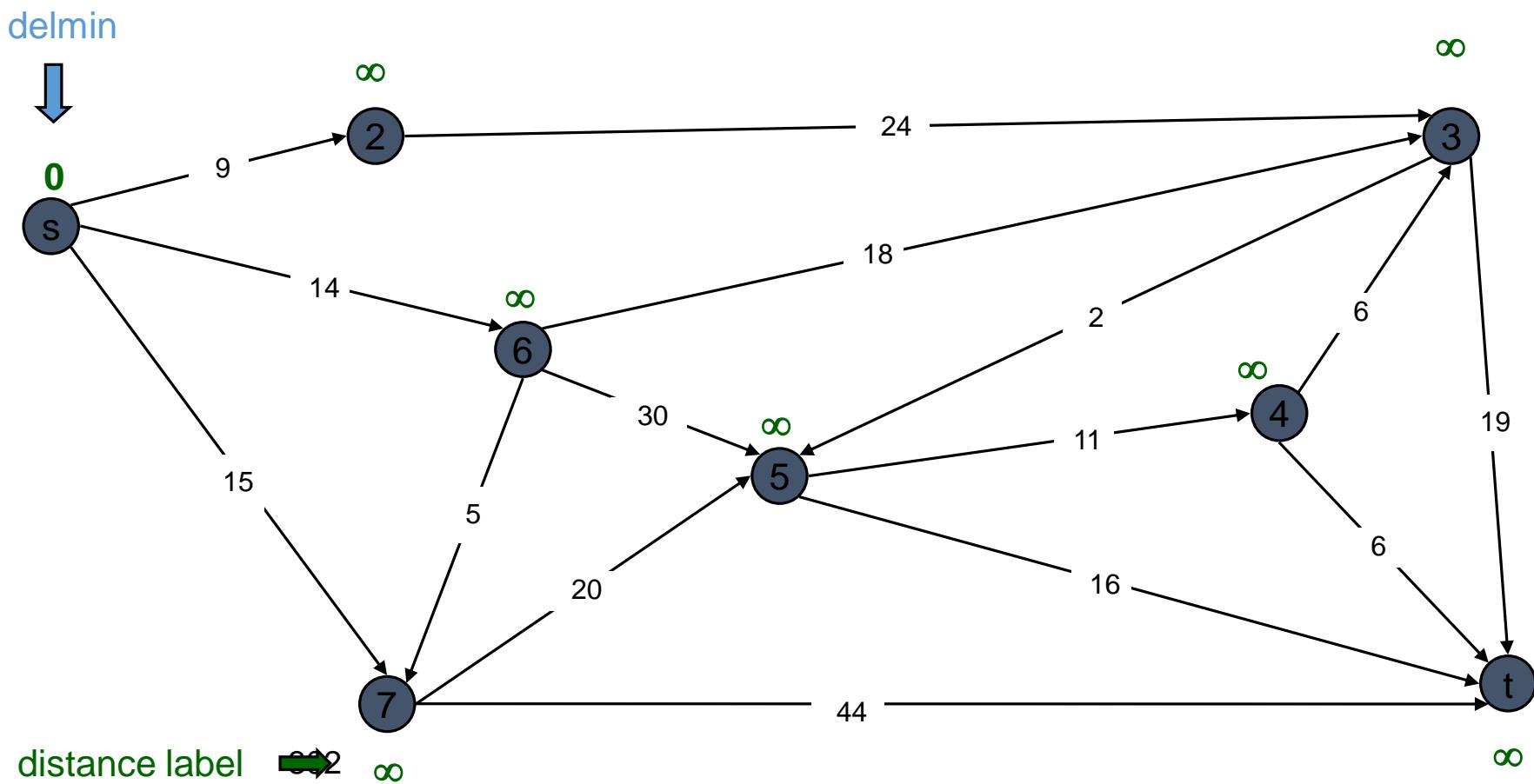
PQ = { s, 2, 3, 4, 5, 6, 7, t }



# Dijkstra's Shortest Path Algorithm

$S = \{ \}$

$PQ = \{ s, 2, 3, 4, 5, 6, 7, t \}$

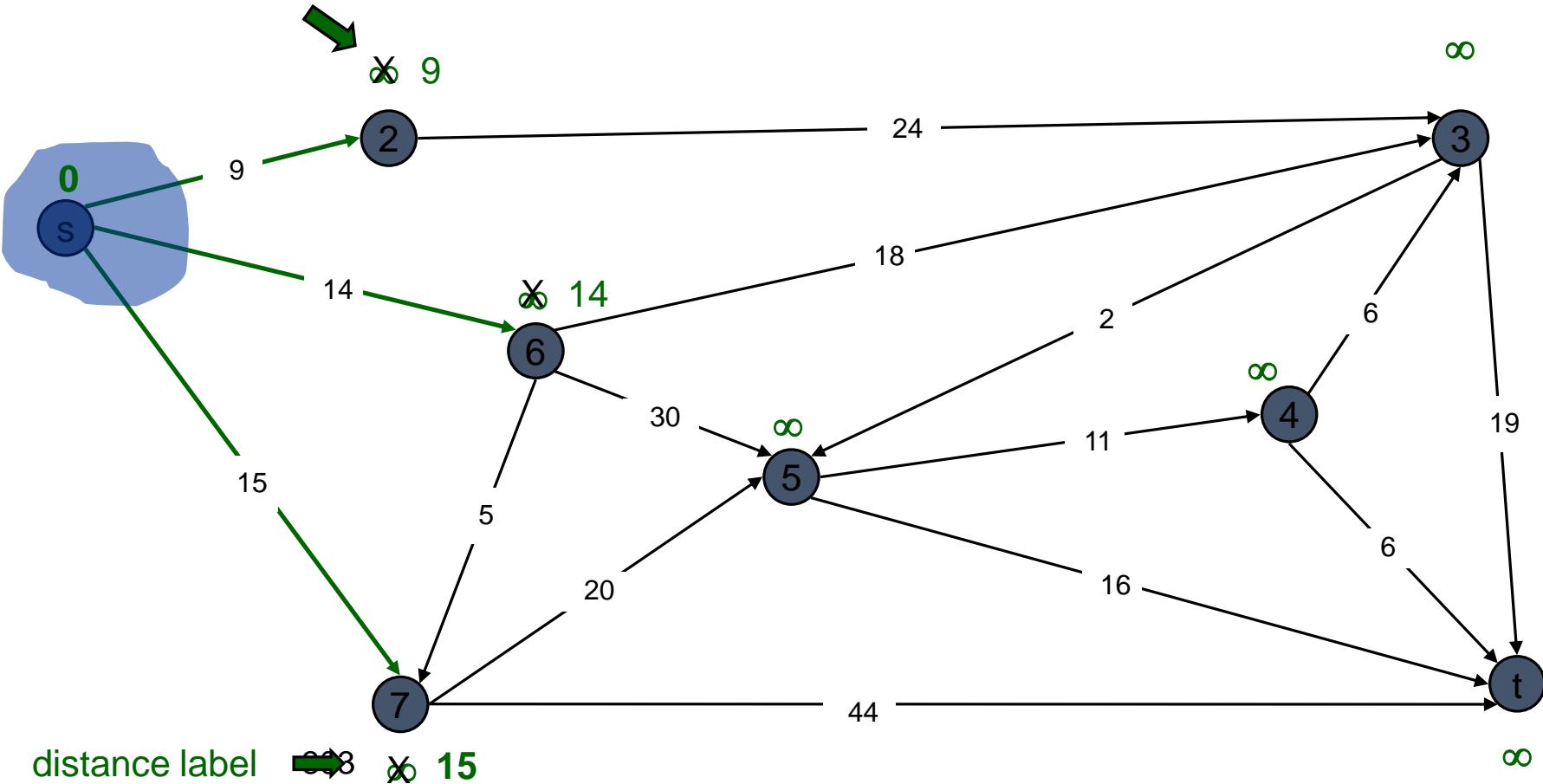


# Dijkstra's Shortest Path Algorithm

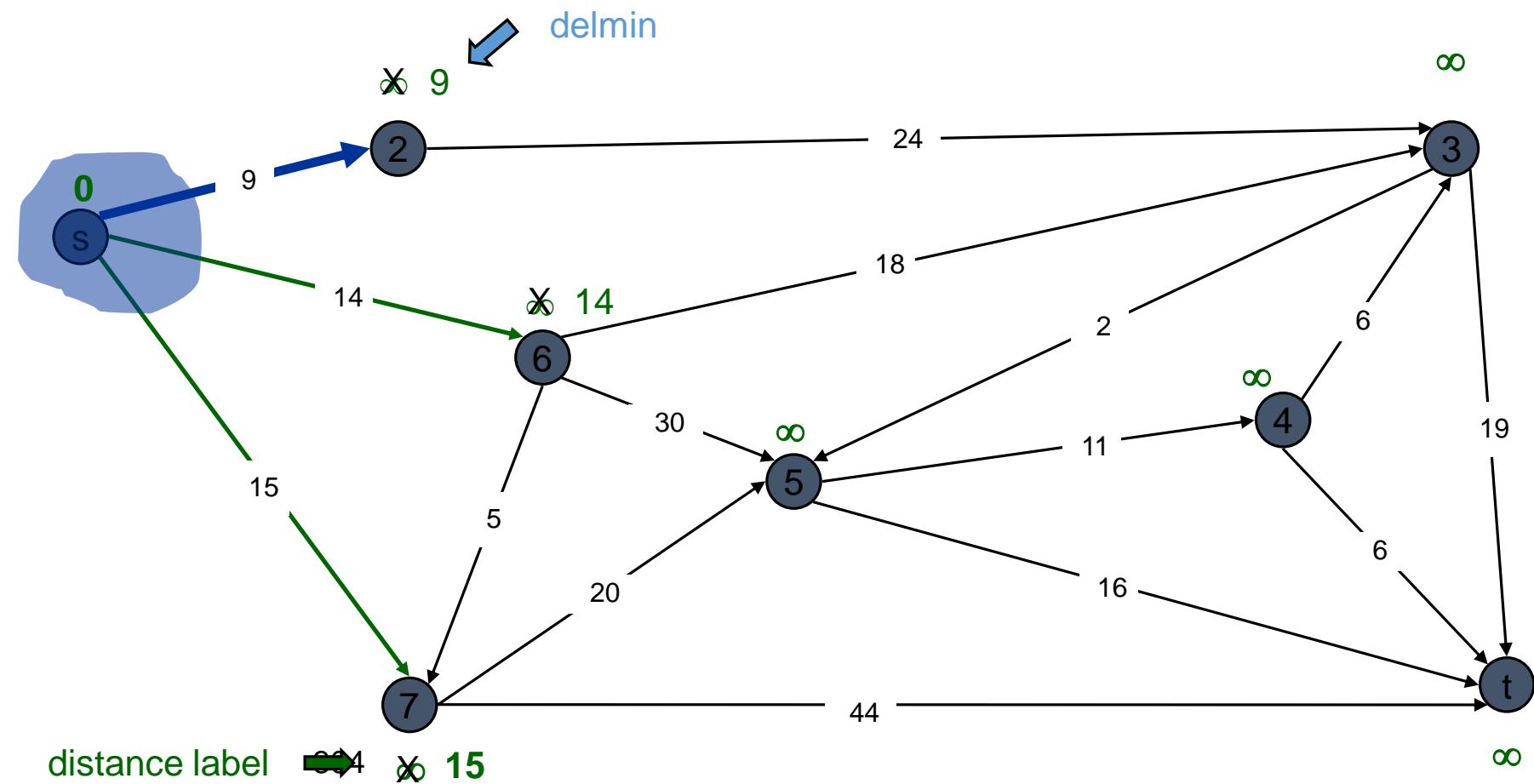
$$S = \{ s \}$$

$$PQ = \{ 2, 3, 4, 5, 6, 7, t \}$$

decrease key



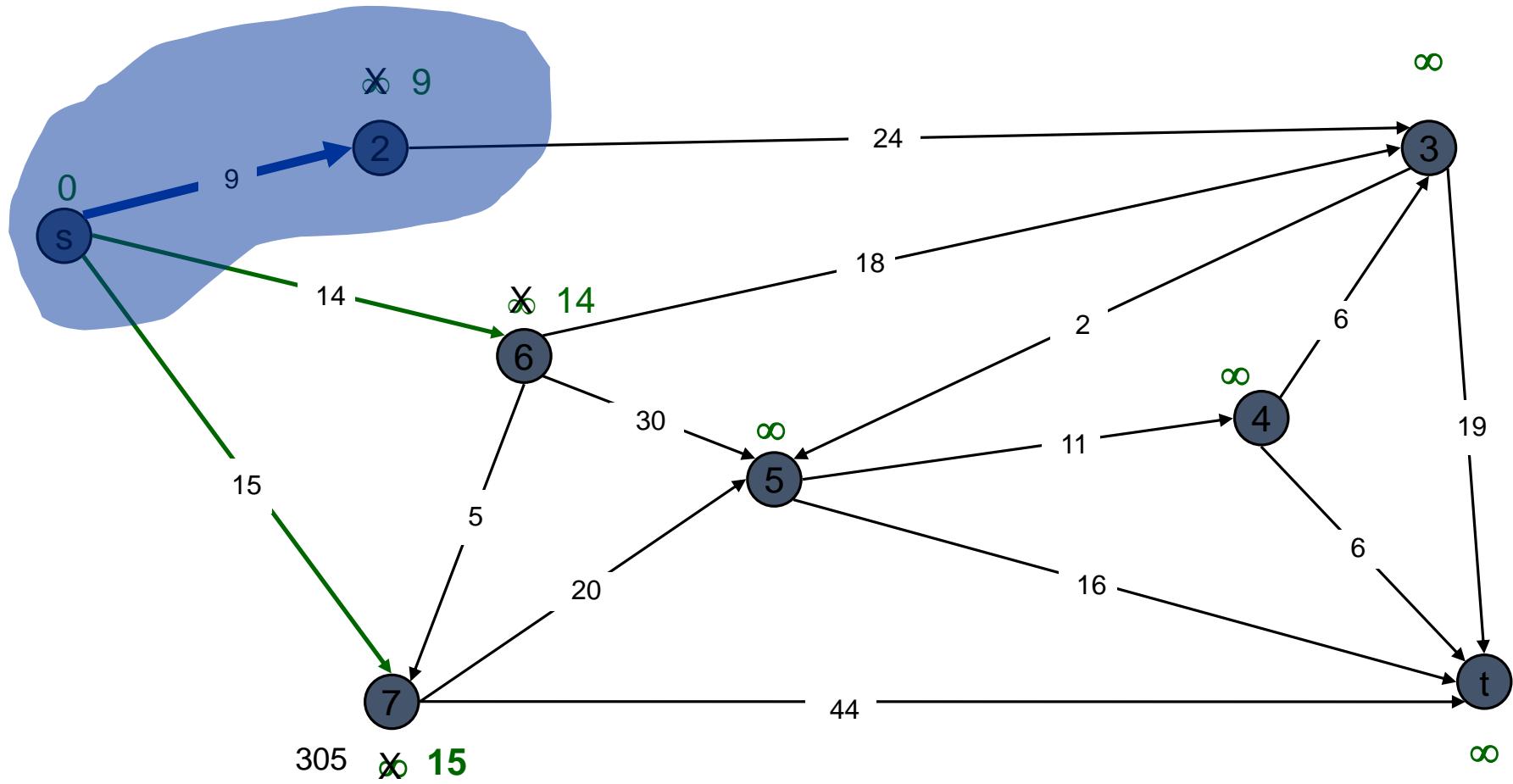
# Dijkstra's Shortest Path Algorithm



# Dijkstra's Shortest Path Algorithm

$S = \{ s, 2 \}$

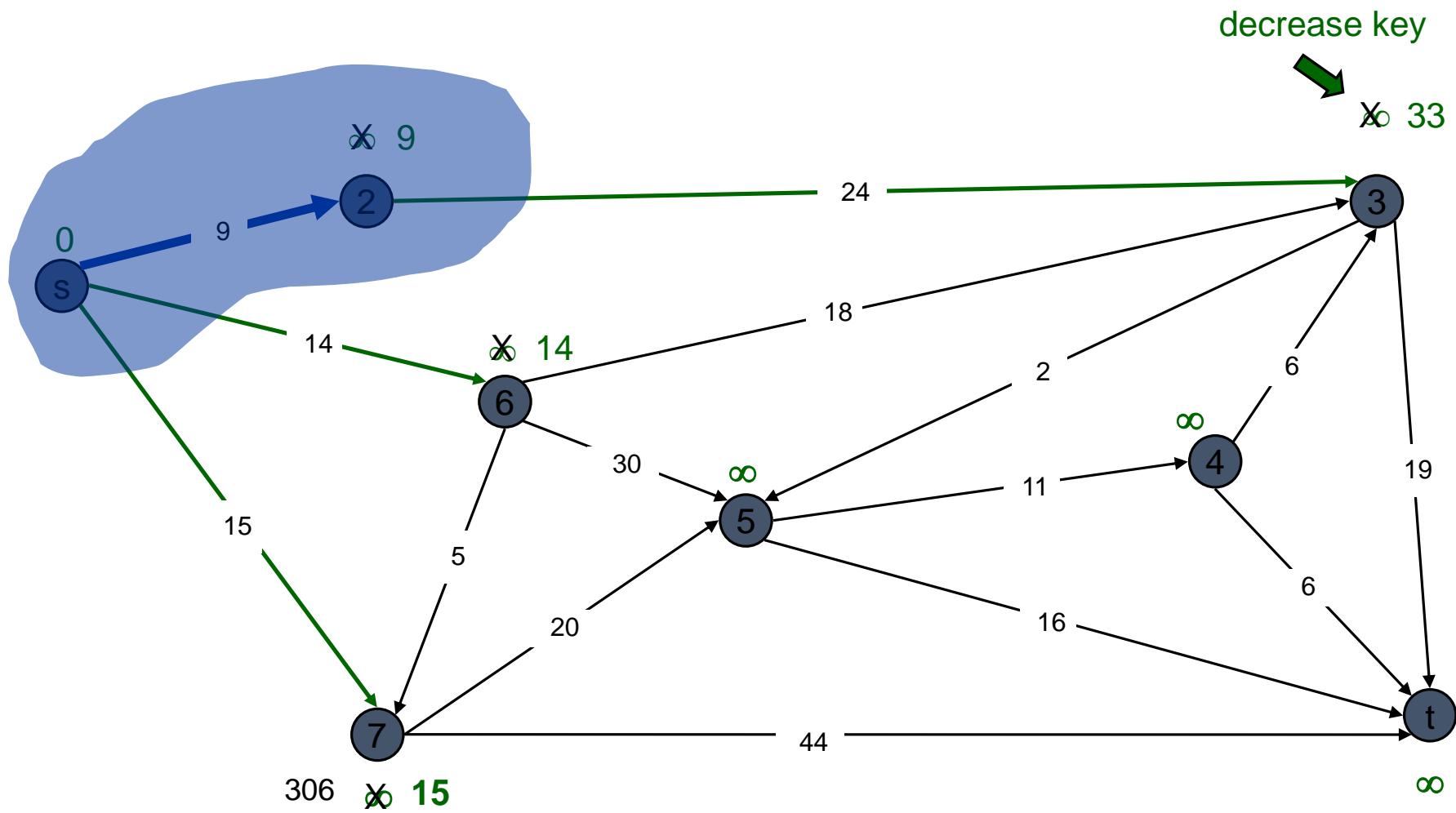
$PQ = \{ 3, 4, 5, 6, 7, t \}$



# Dijkstra's Shortest Path Algorithm

$S = \{ s, 2 \}$

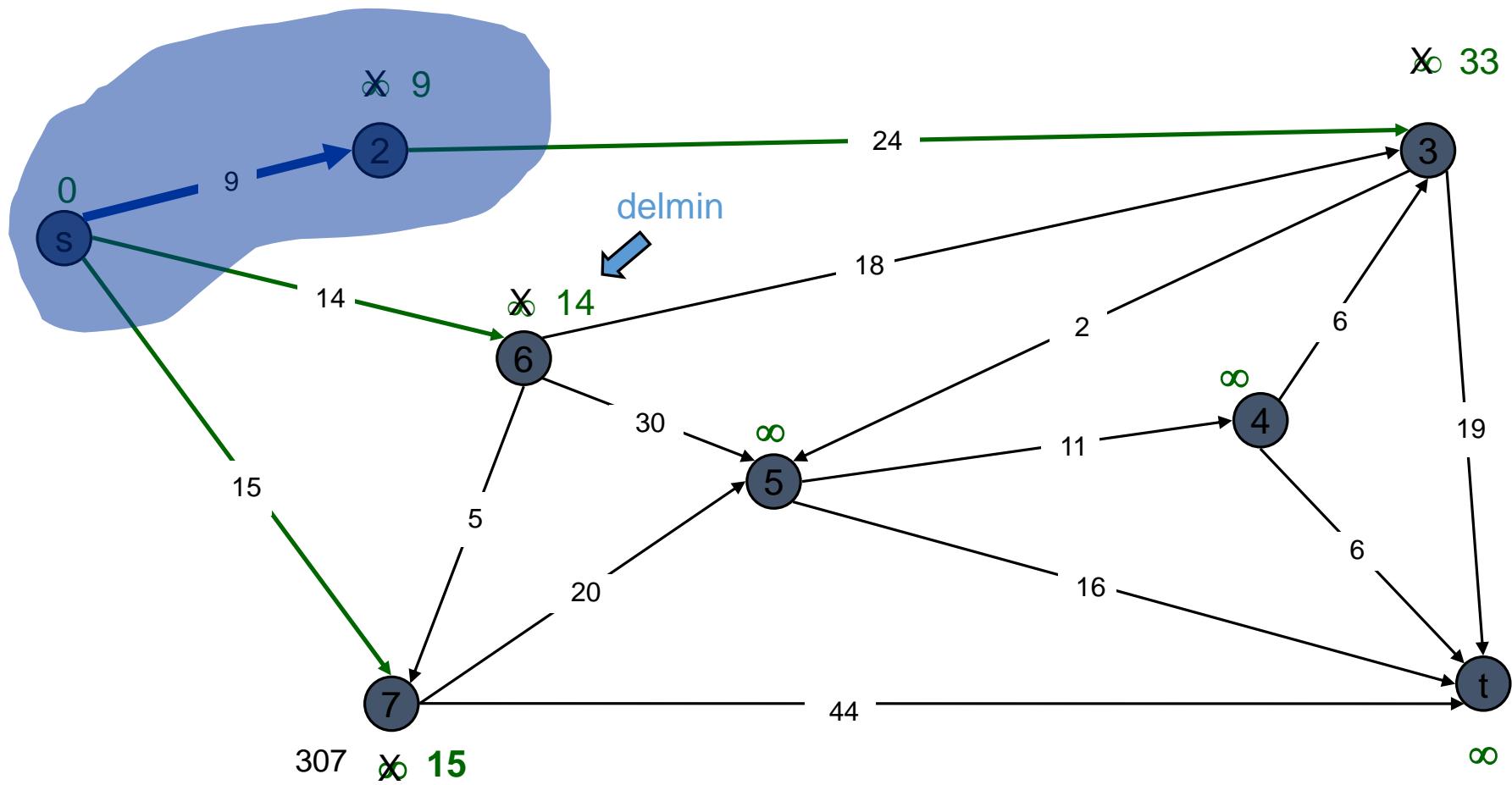
$PQ = \{ 3, 4, 5, 6, 7, t \}$



# Dijkstra's Shortest Path Algorithm

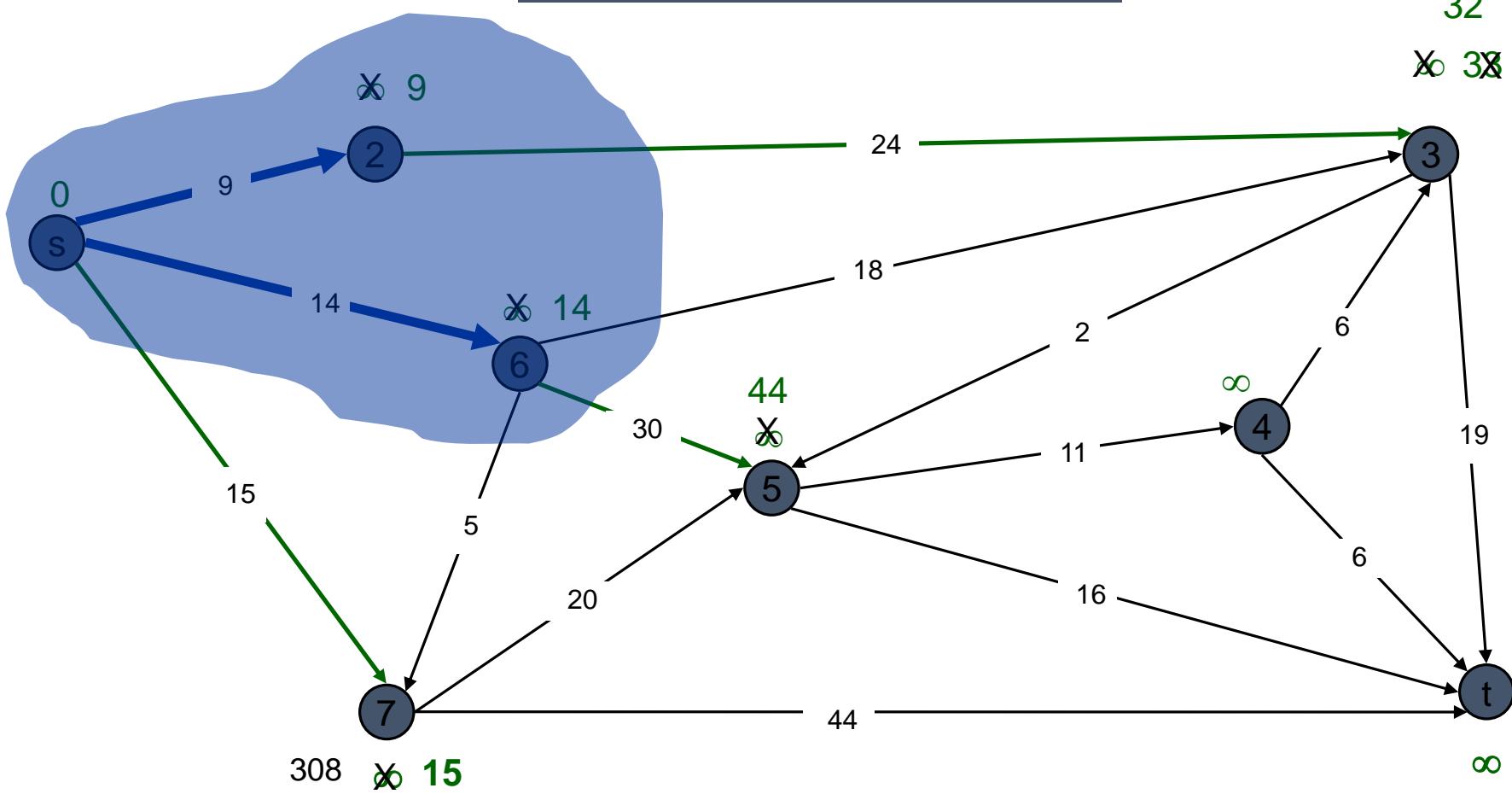
$S = \{ s, 2 \}$

$PQ = \{ 3, 4, 5, 6, 7, t \}$



# Dijkstra's Shortest Path Algorithm

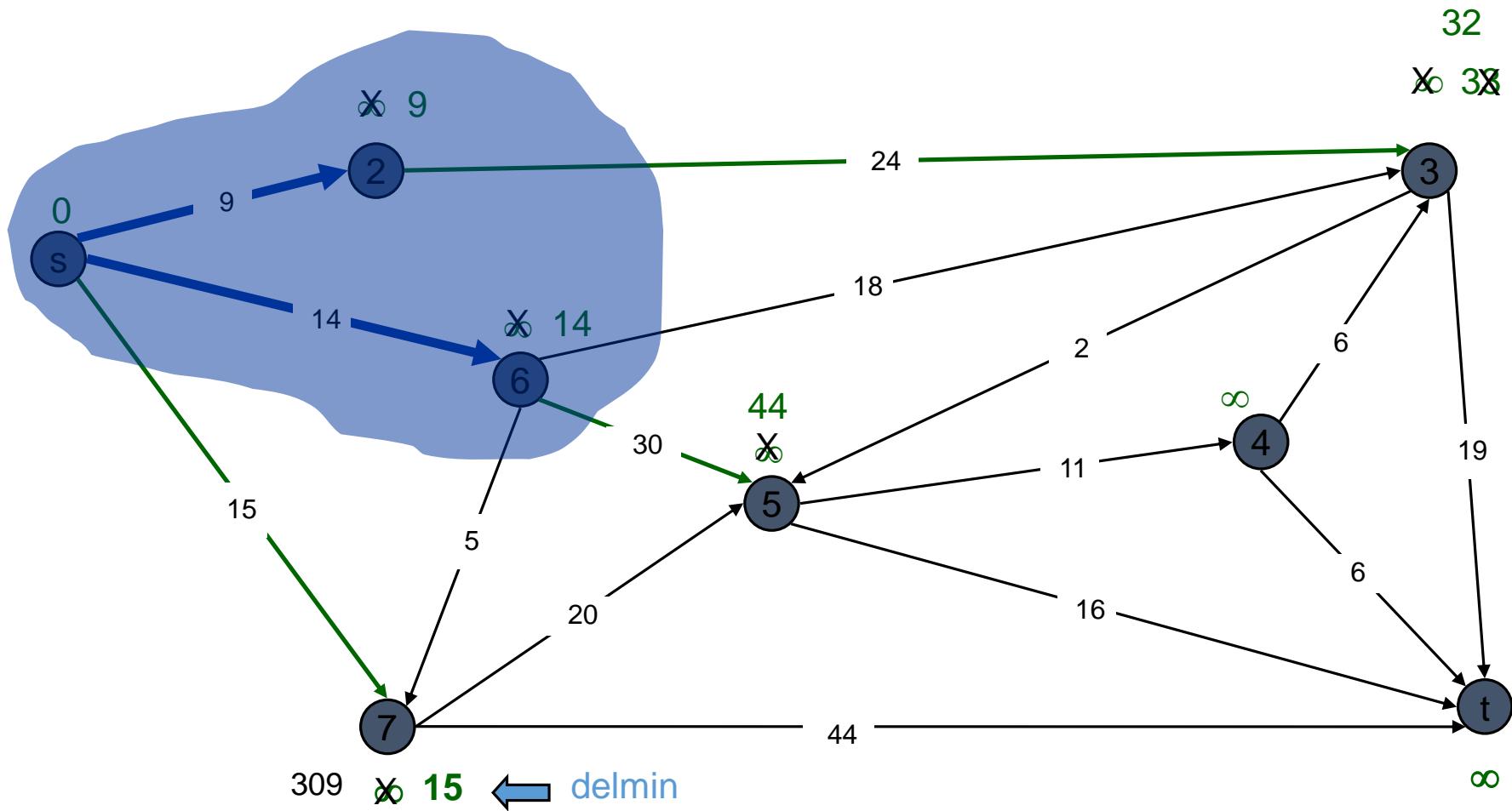
$S = \{ s, 2, 6 \}$   
 $PQ = \{ 3, 4, 5, 7, t \}$



# Dijkstra's Shortest Path Algorithm

$S = \{ s, 2, 6 \}$

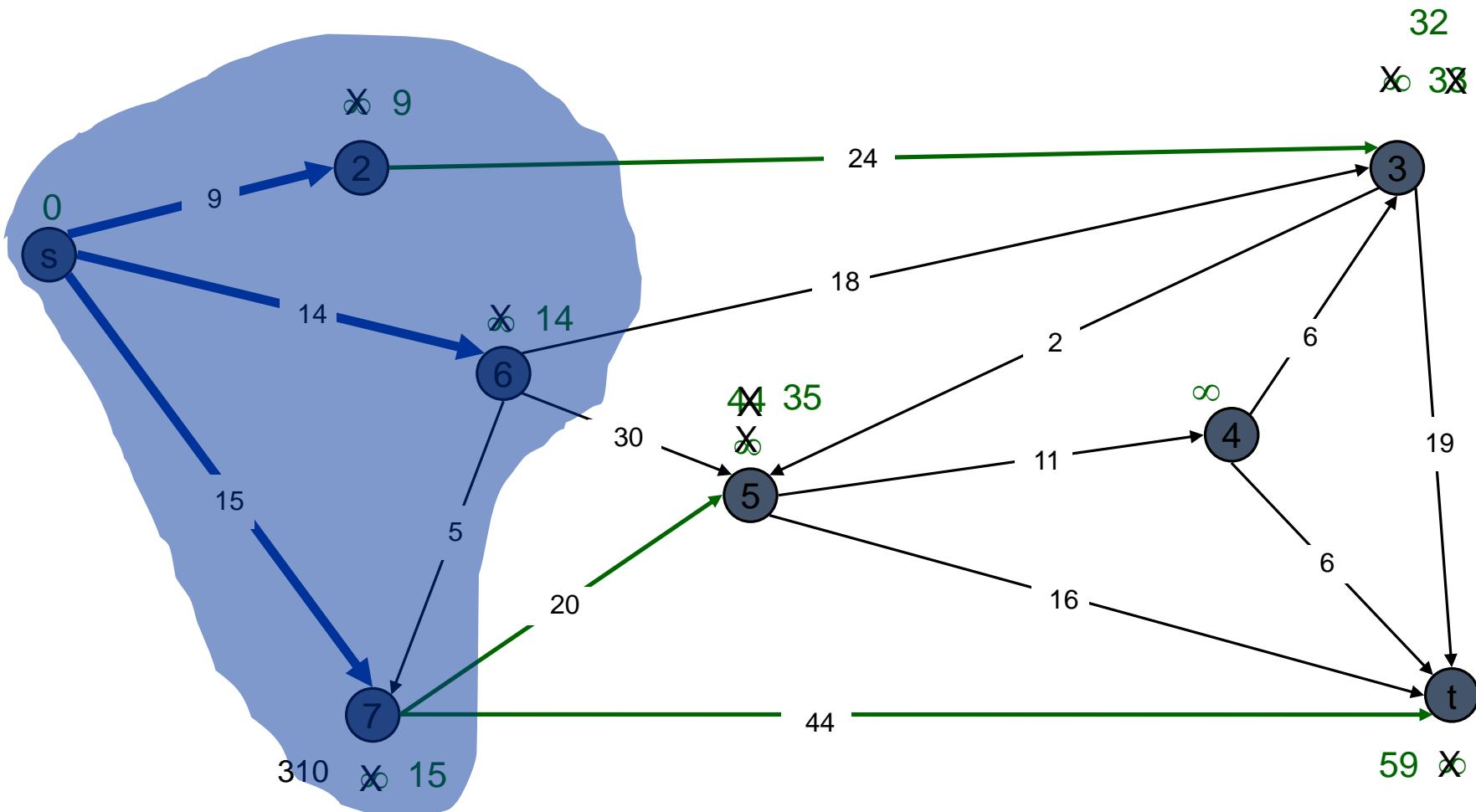
$PQ = \{ 3, 4, 5, 7, t \}$



# Dijkstra's Shortest Path Algorithm

$S = \{ s, 2, 6, 7 \}$

$PQ = \{ 3, 4, 5, t \}$

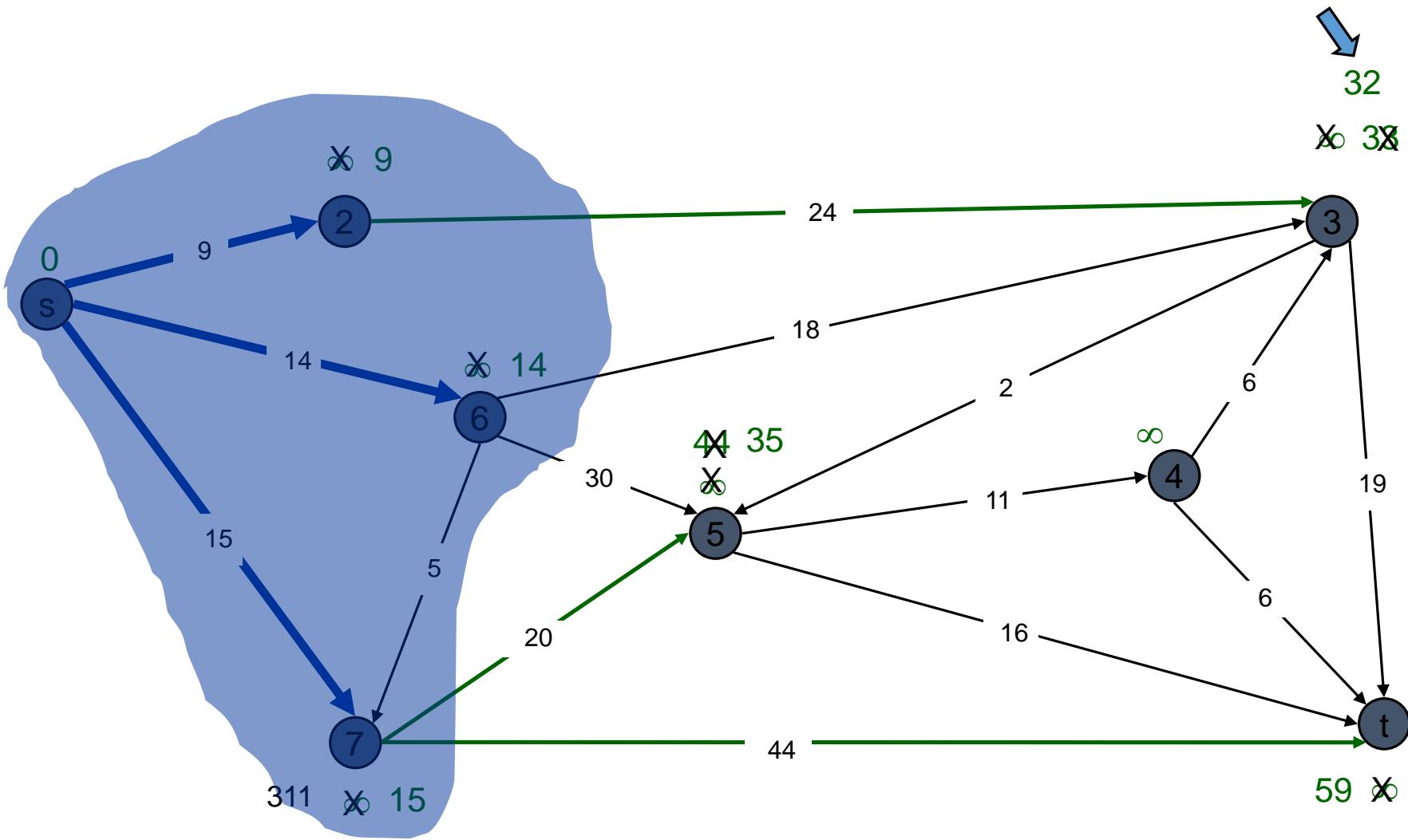


# Dijkstra's Shortest Path Algorithm

$S = \{ s, 2, 6, 7 \}$

$PQ = \{ 3, 4, 5, t \}$

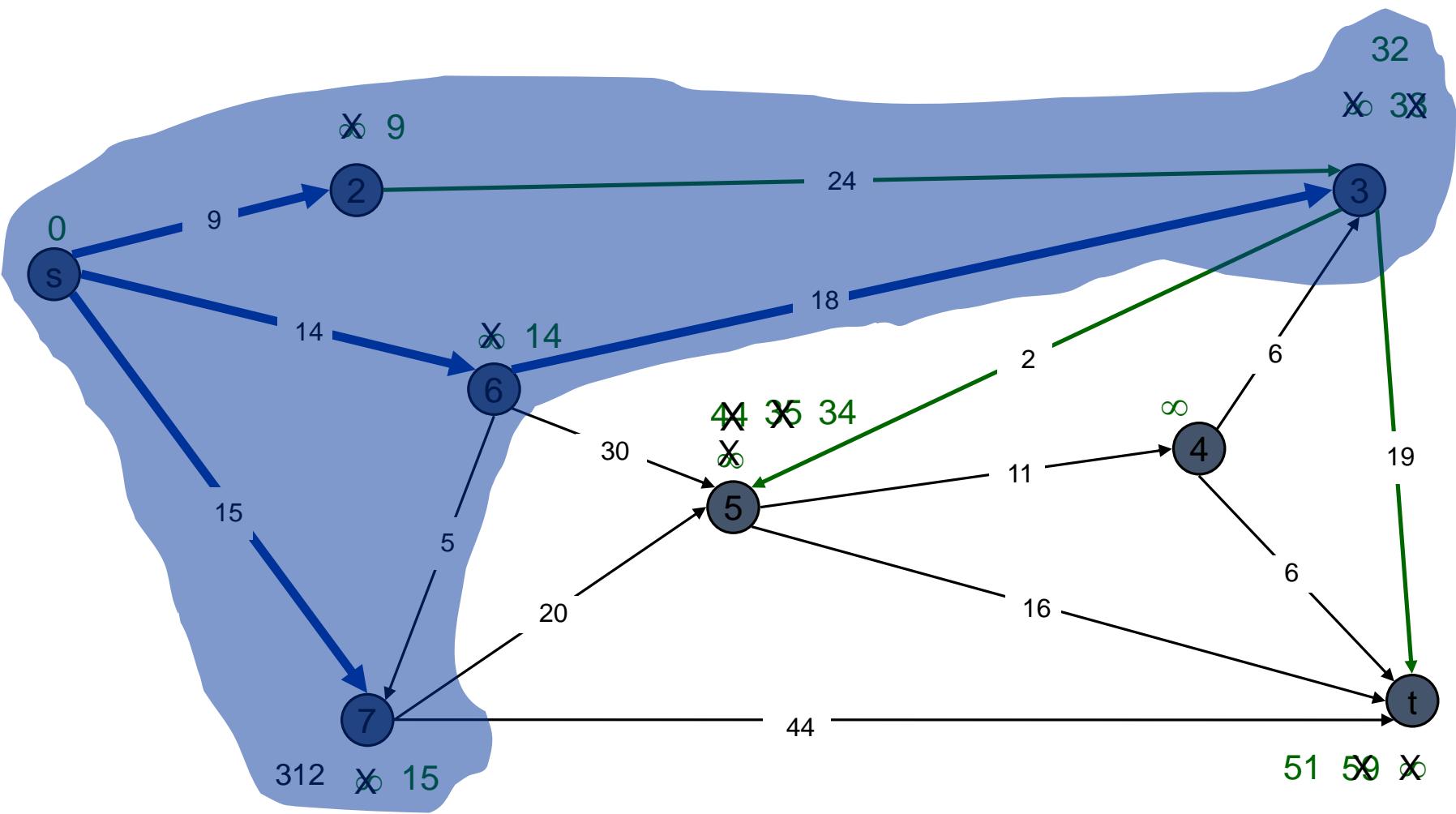
delmin



# Dijkstra's Shortest Path Algorithm

$S = \{ s, 2, 3, 6, 7 \}$

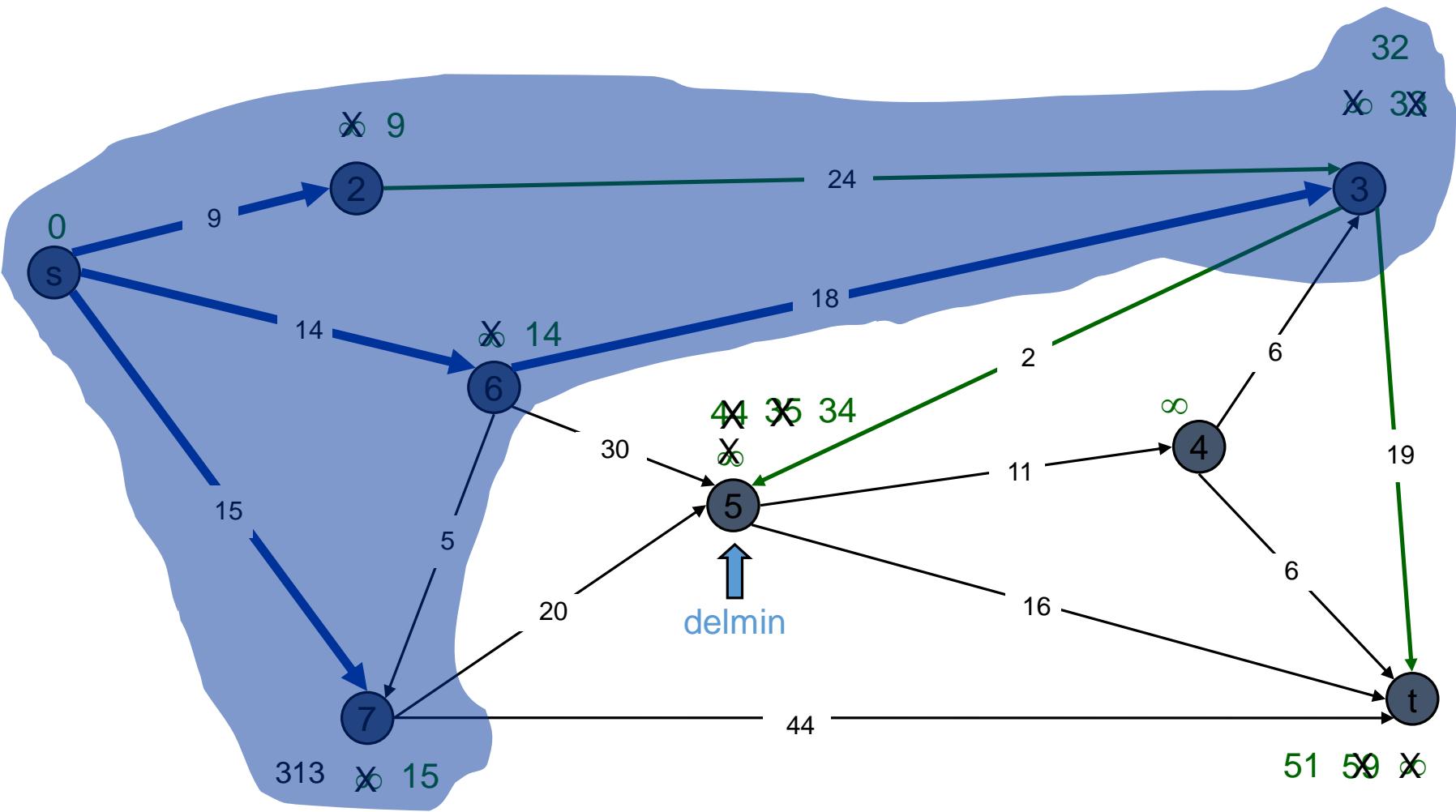
$PQ = \{ 4, 5, t \}$



# Dijkstra's Shortest Path Algorithm

$S = \{ s, 2, 3, 6, 7 \}$

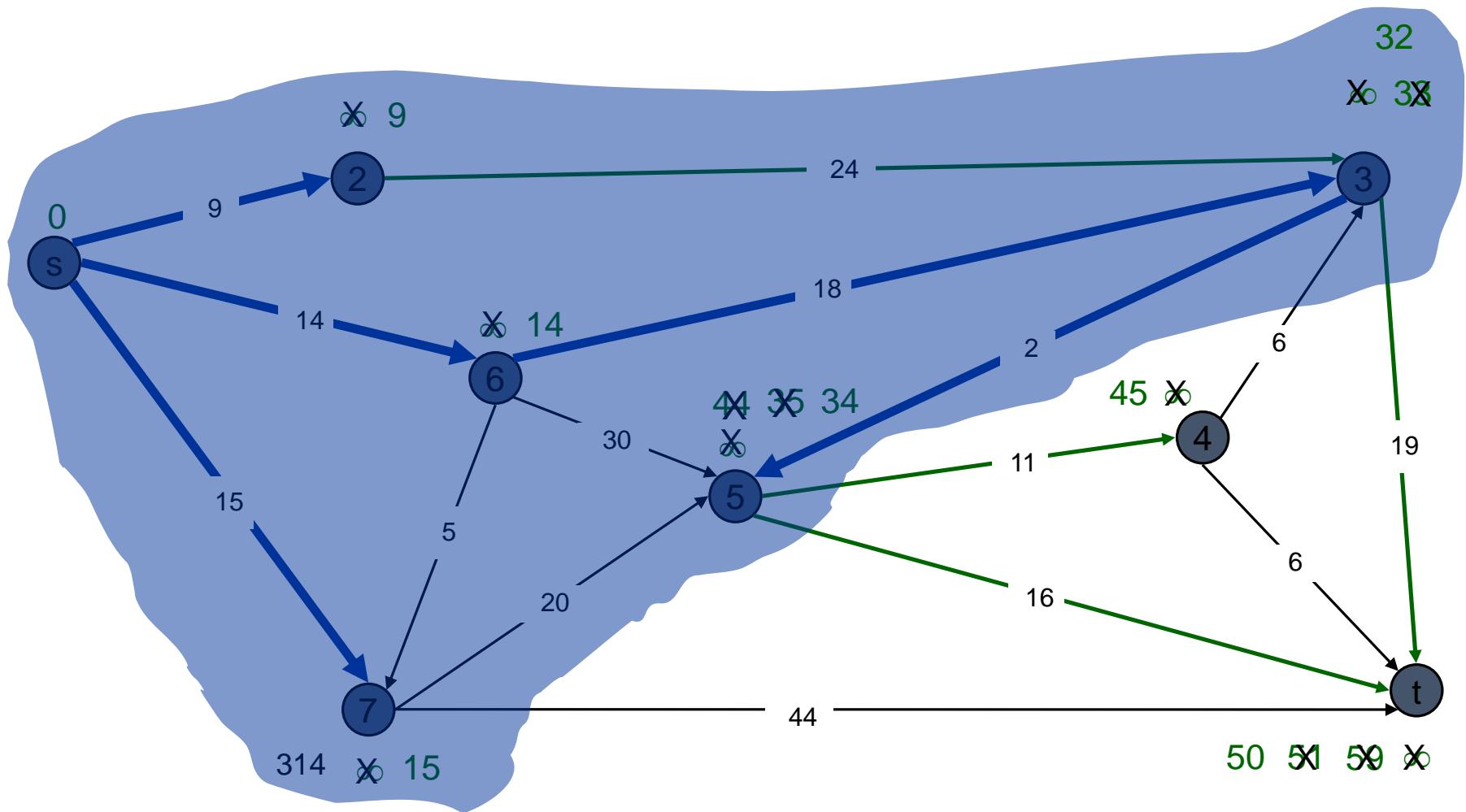
$PQ = \{ 4, 5, t \}$



# Dijkstra's Shortest Path Algorithm

$S = \{ s, 2, 3, 5, 6, 7 \}$

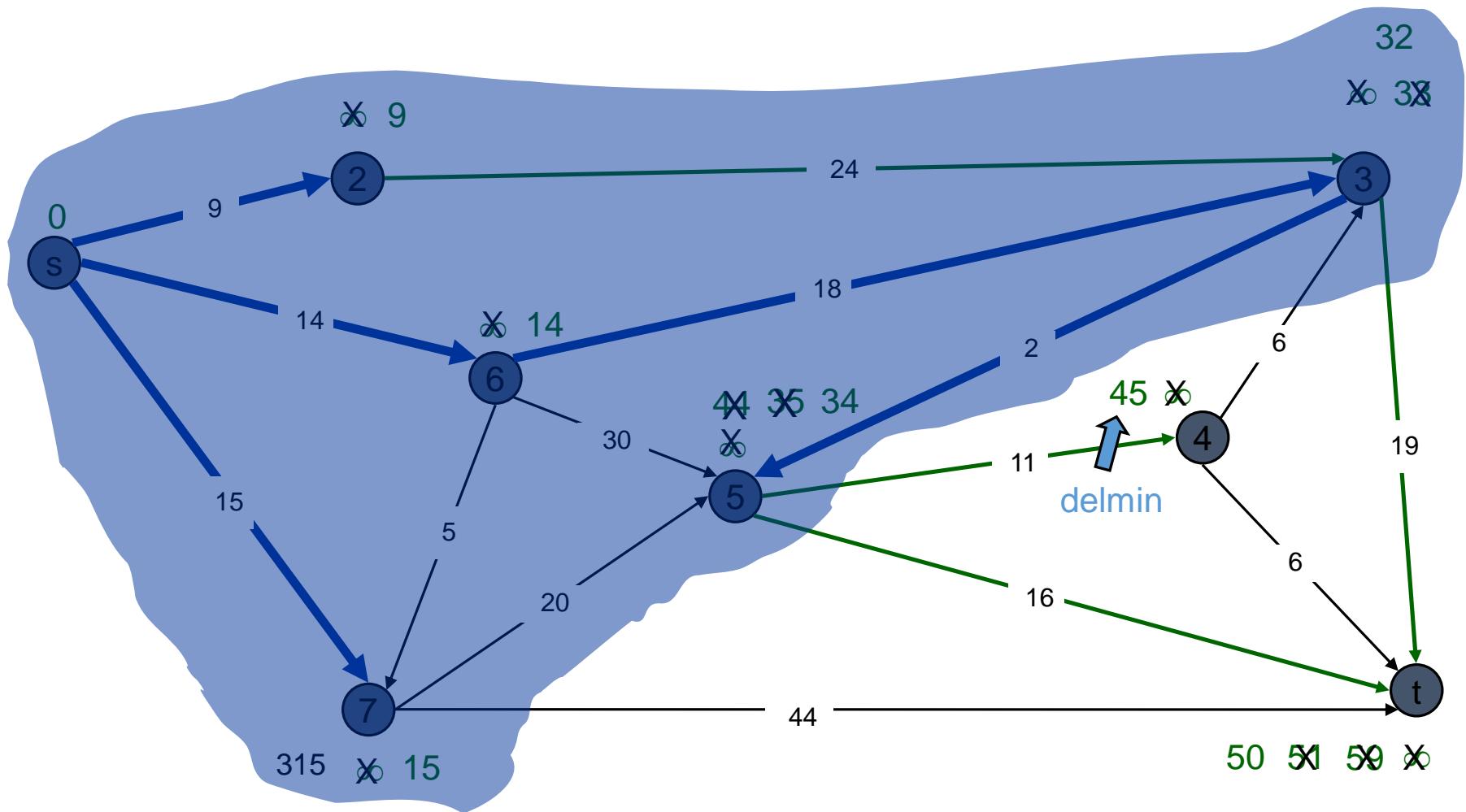
$PQ = \{ 4, t \}$



# Dijkstra's Shortest Path Algorithm

$S = \{ s, 2, 3, 5, 6, 7 \}$

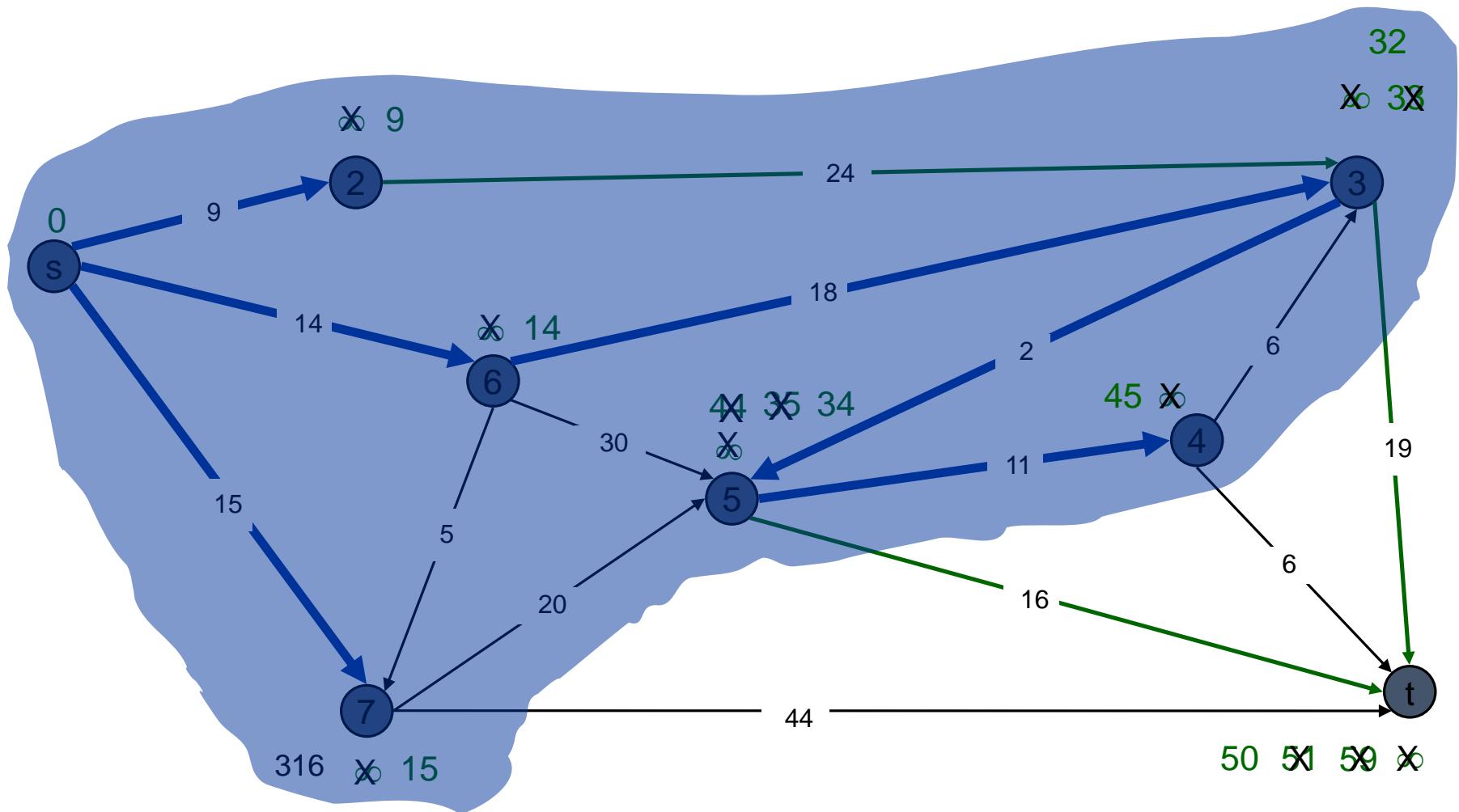
$PQ = \{ 4, t \}$



# Dijkstra's Shortest Path Algorithm

$S = \{ s, 2, 3, 4, 5, 6, 7 \}$

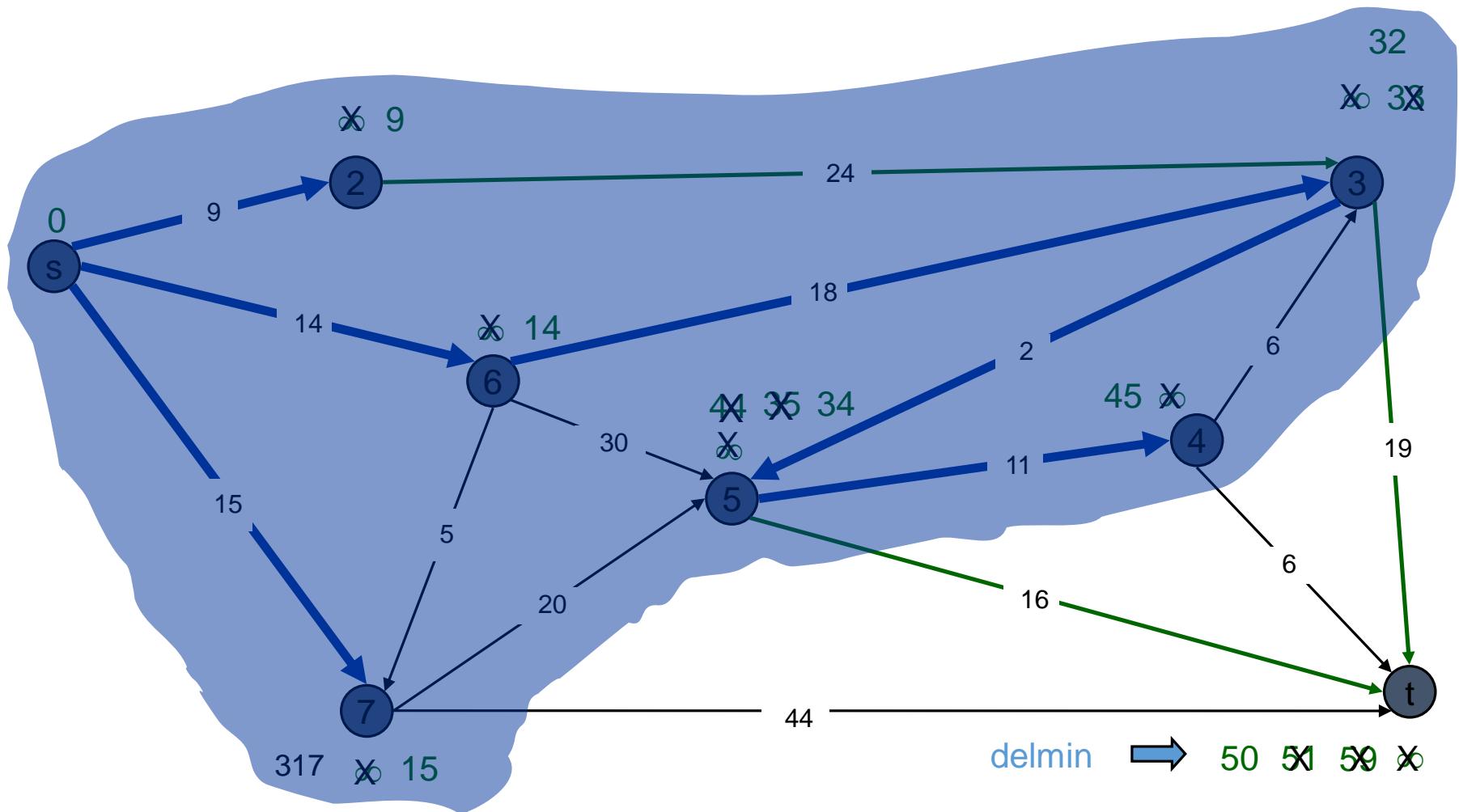
$PQ = \{ t \}$



# Dijkstra's Shortest Path Algorithm

$S = \{ s, 2, 3, 4, 5, 6, 7 \}$

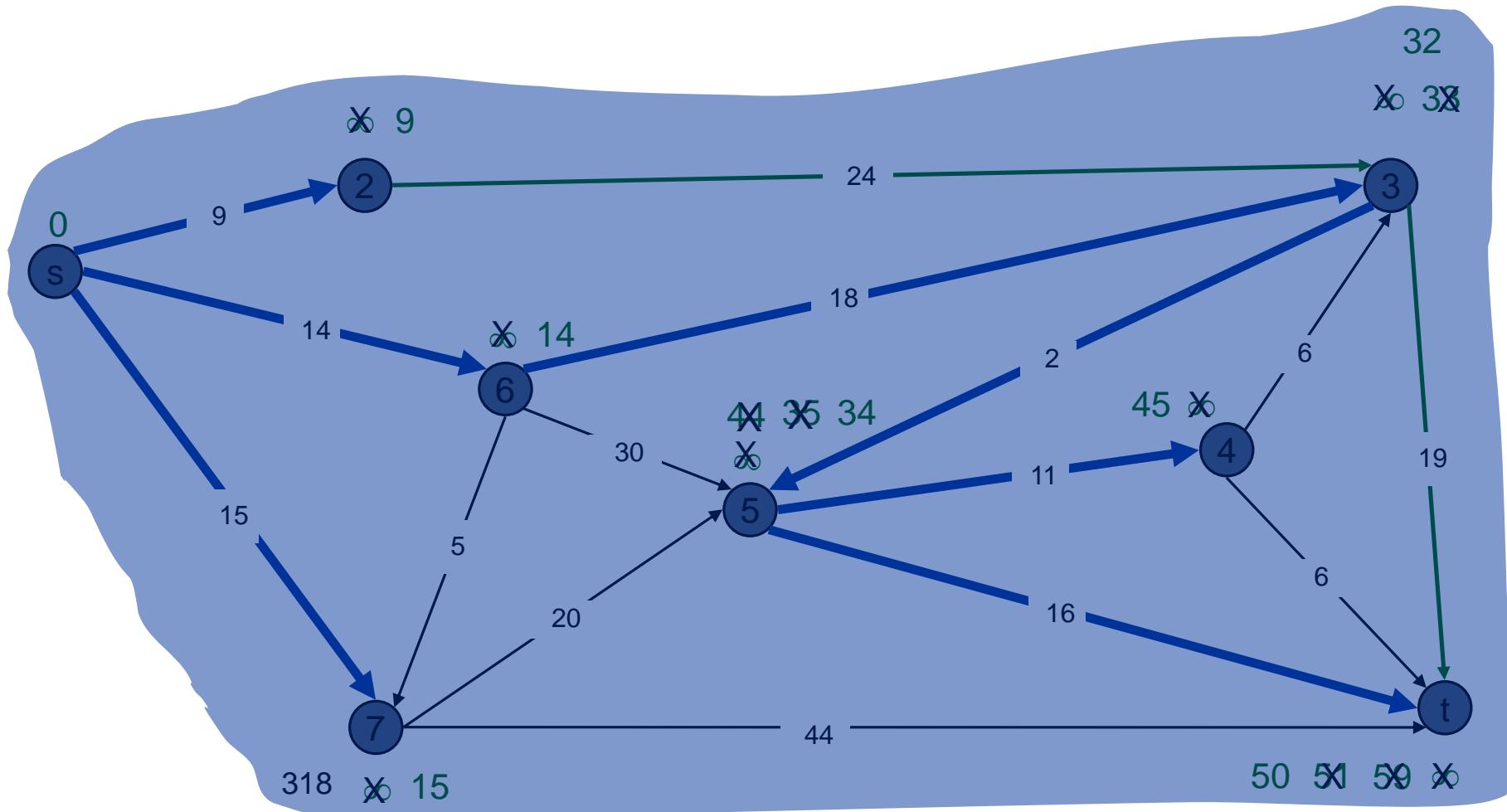
$PQ = \{ t \}$



# Dijkstra's Shortest Path Algorithm

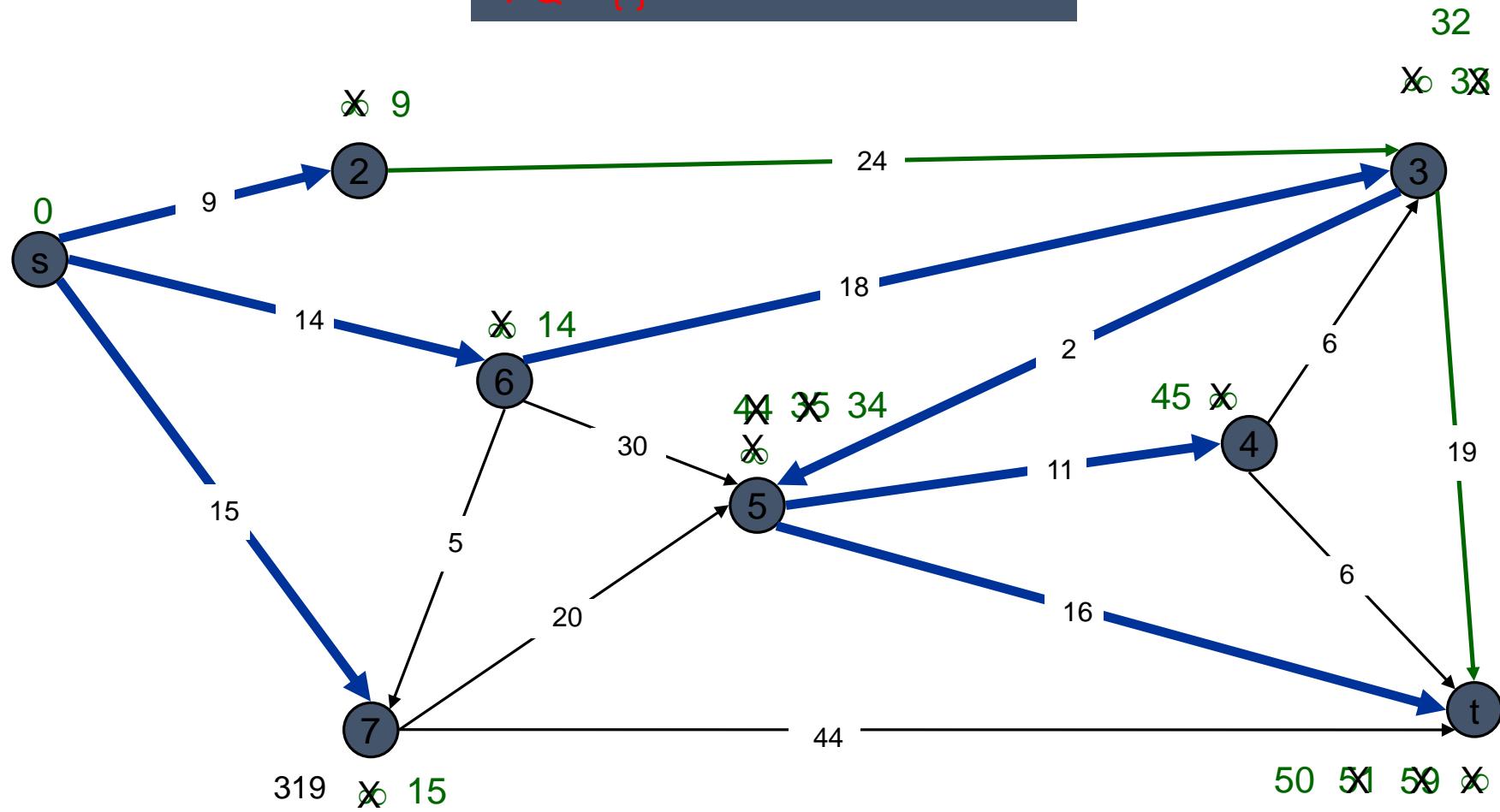
$S = \{ s, 2, 3, 4, 5, 6, 7, t \}$

$PQ = \{ \}$



# Dijkstra's Shortest Path Algorithm

$S = \{ s, 2, 3, 4, 5, 6, 7, t \}$   
 $PQ = \{ \}$



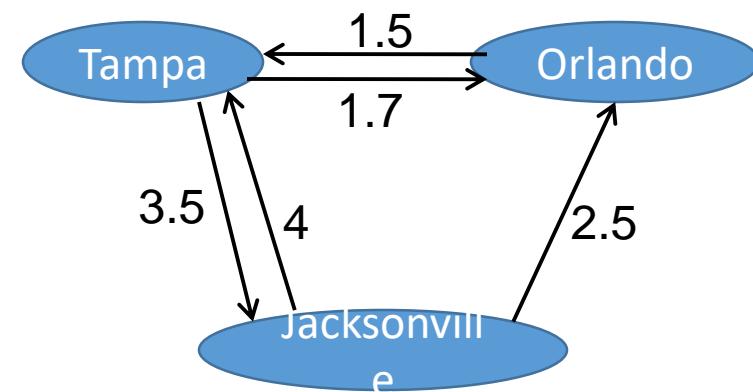
# Floyd's Algorithm

- All pairs shortest path

# Floyd-Warshall Algorithm

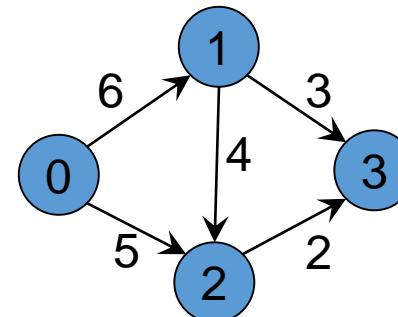
- A weighted, directed graph is a collection of vertices connected by weighted edges (where the weight is some real number).
  - One of the most common examples of a graph in the real world is a road map.
    - Each location is a vertex and each road connecting locations is an edge.
    - We can think of the distance traveled on a road from one location to another as the weight of that edge.

	Tampa	Orlando	Jaxville
Tampa	0	1.7	3.5
Orlando	1.5	0	$\infty$
Jax	4	2.5	0



# Storing a Weighted, Directed Graph

- Adjacency Matrix:
  - Let  $\mathbf{D}$  be an edge-weighted graph in adjacency-matrix form
  - $D(i,j)$  is the weight of edge  $(i, j)$ , or  $\infty$  if there is no such edge.
- Update matrix  $\mathbf{D}$ , with the shortest path *through immediate vertices*.

$$D = \begin{array}{c} & \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} & \left[ \begin{matrix} 0 & 6 & 5 & \infty \\ \infty & 0 & 4 & 3 \\ \infty & \infty & 0 & 2 \\ \infty & \infty & \infty & 0 \end{matrix} \right] \end{array}$$


# Floyd-Warshall Algorithm

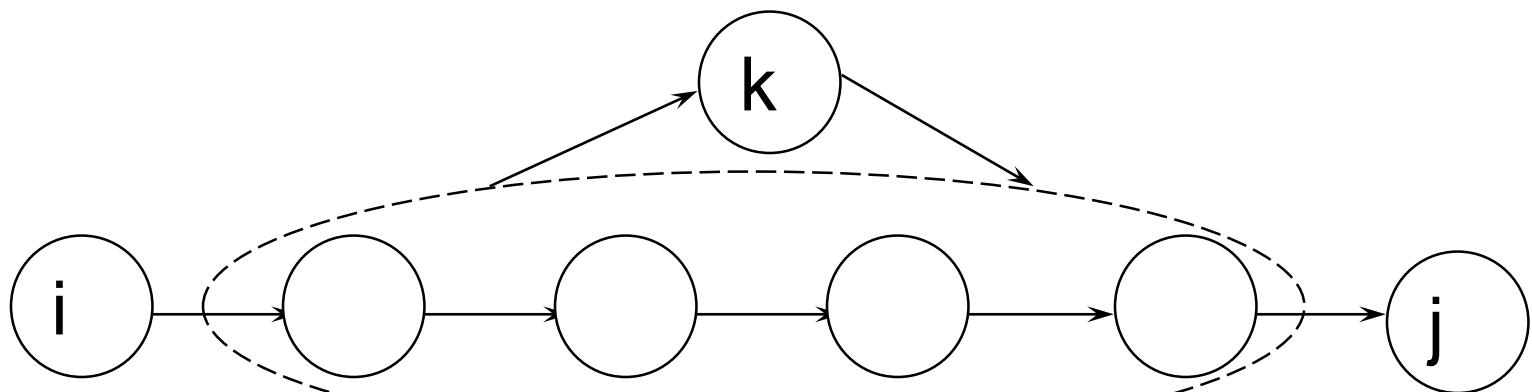
- Given a weighted graph, we want to know the shortest path from one vertex in the graph to another.
  - The Floyd-Warshall algorithm determines the shortest path between all pairs of vertices in a graph.
- What is the difference between Floyd-Warshall and Dijkstra's??

# Floyd-Warshall Algorithm

- If  $V$  is the number of vertices, Dijkstra's runs in  $\Theta(V^2)$ 
  - We could just call Dijkstra  $|V|$  times, passing a different source vertex each time.
  - $\Theta(V \times V^2) = \Theta(V^3)$
  - (Which is the same runtime as the Floyd-Warshall Algorithm)
- BUT, Dijkstra's doesn't work with negative-weight edges.

# Floyd Warshall Algorithm

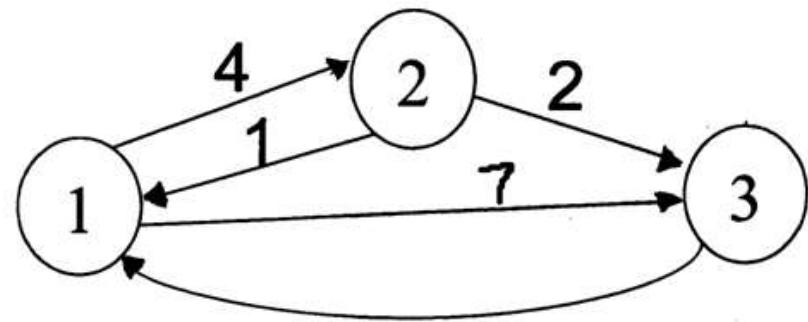
- Let's go over the premise of how Floyd-Warshall algorithm works...
  - Let the vertices in a graph be numbered from 1 ... n.
  - Consider the subset  $\{1, 2, \dots, k\}$  of these n vertices.
  - Imagine finding the shortest path from vertex  $i$  to vertex  $j$  that uses vertices in the set  $\{1, 2, \dots, k\}$  only.
- There are two situations:
  - 1)  $k$  is an intermediate vertex on the shortest path.
  - 2)  $k$  is not an intermediate vertex on the shortest path.



# Floyd Warshall Algorithm - Example

$$D^{(0)} = \begin{bmatrix} \infty & 4 & 7 \\ 1 & \infty & 2 \\ 6 & \infty & \infty \end{bmatrix}$$

Original weights.



$$D^{(1)} = \begin{bmatrix} \infty & 4 & 7 \\ 1 & \infty & 2 \\ 6 & 10 & \infty \end{bmatrix}$$

Consider Vertex 1:  
 $D(3,2) = D(3,1) + D(1,2)$

$$D^{(2)} = \begin{bmatrix} \infty & 4 & 6 \\ 1 & \infty & 2 \\ 6 & 10 & \infty \end{bmatrix}$$

Consider Vertex 2:  
 $D(1,3) = D(1,2) + D(2,3)$

$$D^{(3)} = \begin{bmatrix} \infty & 4 & 6 \\ 1 & \infty & 2 \\ 6 & 10 & \infty \end{bmatrix}$$

Consider Vertex 3:  
Nothing changes.

# Floyd Warshall Algorithm

- Looking at this example, we can come up with the following algorithm:
  - Let D store the matrix with the initial graph edge information initially, and update D with the calculated shortest paths.

```
For k=1 to n {  
    For i=1 to n {  
        For j=1 to n  
            D[i,j] =  
min(D[i,j],D[i,k]+D[k,j])  
    }  
}
```

- The final D matrix will store all the shortest paths.

# Floyd Warshall – Path Reconstruction

- The path matrix will store the last vertex visited on the path from i to j.
  - So  $\text{path}[i][j] = k$  means that in the shortest path from vertex i to vertex j, the LAST vertex on that path before you get to vertex j is k.
- Based on this definition, we must initialize the path matrix as follows:
  - $\text{path}[i][j] = i$  if  $i \neq j$  and there exists an edge from i to j
  - $= \text{NIL}$  otherwise
- The reasoning is as follows:
  - If you want to reconstruct the path at this point of the algorithm when you aren't allowed to visit intermediate vertices, the previous vertex visited MUST be the source vertex i.
  - NIL is used to indicate the absence of a path.

# Floyd Warshall – Path Reconstruction

- Before you run Floyd's, you initialize your distance matrix  $D$  and path matrix  $P$  to indicate the use of no immediate vertices.
  - (Thus, you are only allowed to traverse direct paths between vertices.)
- Then, at each step of Floyd's, you essentially find out whether or not using vertex  $k$  will *improve* an estimate between the distances between vertex  $i$  and vertex  $j$ .
- If it *does improve* the estimate here's what you need to record:
  - 1) record the new shortest path weight between  $i$  and  $j$
  - 2) record the fact that the shortest path between  $i$  and  $j$  goes through  $k$

# Floyd Warshall – Path Reconstruction

- If it ***does improve*** the estimate here's what you need to record:
  - 1) record the new shortest path weight between i and j
    - We don't need to change our path and we do not update the path matrix
  - 2) record the fact that the shortest path between i and j goes through k
    - We want to store the last vertex from the shortest path from vertex k to vertex j. *This will NOT necessarily be k, but rather, it will be path[k][j].*

This gives us the following update to our algorithm:

```
if (D[i][k]+D[k][j] < D[i][j]) { // Update is necessary to use k as  
    intermediate vertex
```

```
    D[i][j] = D[i][k]+D[k][j];
```

```
    path[i][j] = path[k][j];
```

```
}
```

# Path Reconstruction

- Now, once this path matrix is computed, we have all the information necessary to reconstruct the path.
  - Consider the following path matrix (indexed from 1 to 5 instead of 0 to 4):

<b>NIL</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>1</b>
<b>4</b>	<b>NIL</b>	<b>4</b>	<b>2</b>	<b>1</b>
<b>4</b>	<b>3</b>	<b>NIL</b>	<b>2</b>	<b>1</b>
<b>4</b>	<b>3</b>	<b>4</b>	<b>NIL</b>	<b>1</b>
<b>4</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>NIL</b>

- Reconstruct the path from vertex 1 to vertex 2:
  - First look at path[1][2] = 3. This signifies that on the path from 1 to 2, 3 is the last vertex visited before 2.
    - Thus, the path is now, 1...3->2.
  - Now, look at path[1][3], this stores a 4. Thus, we find the last vertex visited on the path from 1 to 3 is 4.
  - So, our path now looks like 1...4->3->2. So, we must now look at path[1][4]. This stores a 5,
  - thus, we know our path is 1...5->4->3->2. When we finally look at path[1][5], we find 1,
  - which means our path really is 1->5->4->3->2.

# Transitive Closure

- Computing a transitive closure of a graph gives you complete information about which vertices are connected to which other vertices.
- Input:
  - Un-weighted graph  $G$ :  $W[i][j] = 1$ , if  $(i,j) \in E$ ,  $W[i][j] = 0$  otherwise.
- Output:
  - $T[i][j] = 1$ , if there is a path from  $i$  to  $j$  in  $G$ ,  $T[i][j] = 0$  otherwise.
- Algorithm:
  - Just run Floyd-Warshall with weights 1, and make  $T[i][j] = 1$ , whenever  $D[i][j] < \infty$ .
  - More efficient: use only Boolean operators

# Transitive Closure

**Transitive-Closure** ( $W[1..n][1..n]$ )

```
01 T  $\leftarrow W$  //  $T^{(0)}$ 
02 for  $k \leftarrow 1$  to  $n$  do // compute  $T^{(k)}$ 
03     for  $i \leftarrow 1$  to  $n$  do
04         for  $j \leftarrow 1$  to  $n$  do
05              $T[i][j] \leftarrow T[i][j] \vee (T[i][k] \wedge T[k][j])$ 
06 return  $T$ 
```

- This is the SAME as the other Floyd-Warshall Algorithm, except for when we find a non-infinity estimate, we simply add an edge to the transitive closure graph.
- Every round we build off the previous paths reached.
  - After iterating through all vertices being intermediate vertices, we have tried to connect all pairs of vertices  $i$  and  $j$  through all intermediate vertices  $k$ .