浙江大学
ZHEJIANG UNIVERSITY

第九章 空间网络模型与查询

陶煜波
计算机科学与技术学院

第八章 空间查询处理与优化回顾

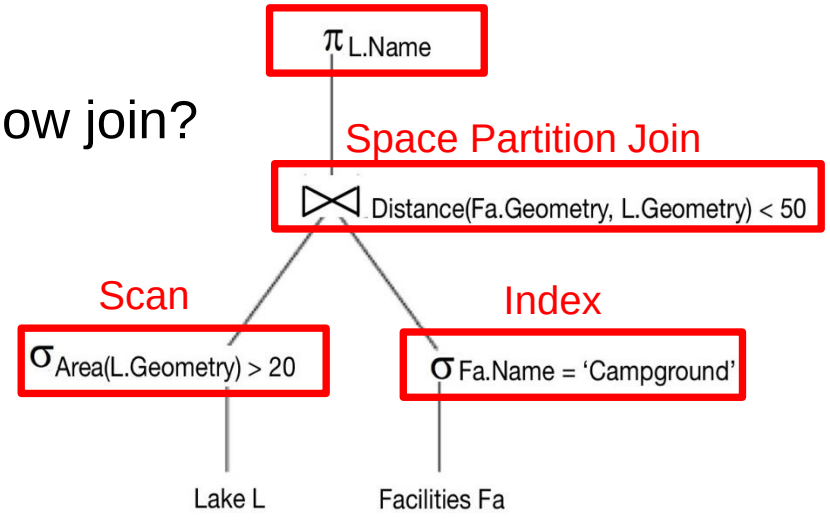
- Query processing and optimization (QPO)
  - Translates SQL Queries to execution plan
- Three key concepts in QPO
  - Building blocks
    - Decompose SQL query into building blocks, e.g., select, join, sort
  - Query processing strategies for building blocks
    - A few processing strategies for each building block
    - E.g. a point query – scan vs. index
  - Query optimization
    - For each building block of a given query, DBMS QPO tries to choose “most efficient” strategy given database parameters
      - A fixed priority scheme
      - Rule based approach
      - Cost-based model

第八章 空间查询处理与优化回顾

- Building blocks and strategies
  - Point Query
    - Linear search, binary search with Z-Curve, Indexed search with R-Tree
  - Range Query
    - Linear search, binary search (+scan) with Z-Curve, Indexed search with R-Tree
  - Nearest Neighbor
    - Two phase approach, single phase approach
  - Spatial Join
    - Nested loop, nested loop with one spatial index, space partitioning, tree matching

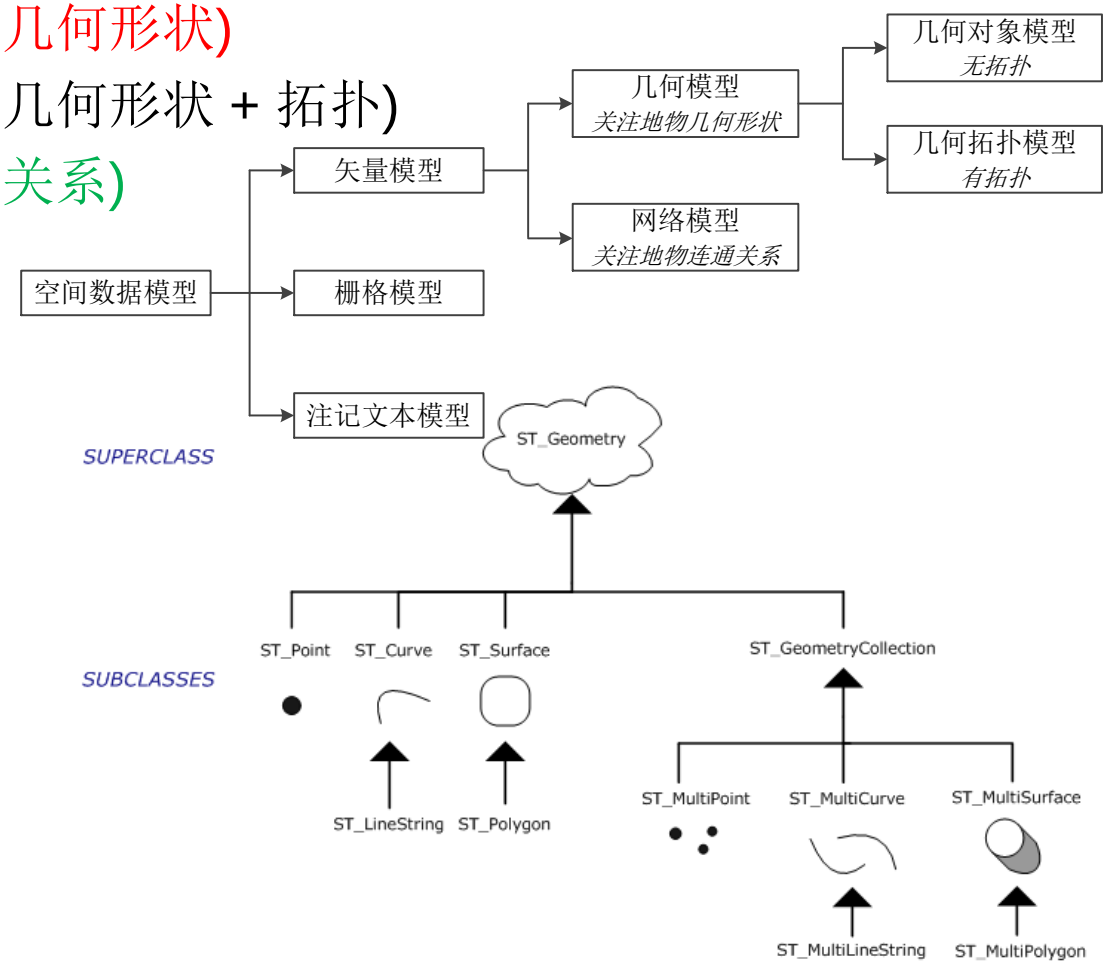
第八章 空间查询处理与优化回顾

- QPO process steps include
  - Creation of a query tree for the SQL query
  - Choice of strategies to process each node in query tree
  - Ordering the nodes for execution
- Key ideas for SDBMS include
  - Filter-Refine paradigm to reduce complexity
  - New building blocks and strategies for spatial queries
  - CPU cost is higher
    - Push down spatial selection below join?
    - Reorder join operations



几何对象模型

- 空间数据模型分类
  - 矢量模型
    - 几何对象模型 (地物几何形状)
    - 几何拓扑模型 (地物几何形状 + 拓扑)
    - 网络模型 (地物连通关系)
  - 栅格模型
  - 注记文本模型
- 几何对象模型
  - 数据 + 方法



空间数据库

- 空间数据库 = 对象关系/关系数据库 + 空间扩展
  - Oracle + Oracle Spatial
  - SQL Server + SQL Server Spatial
  - PostgreSQL + PostGIS
  - MySQL + MySQL Spatial
  - SQLite + SQLite Spatialite
- PostGIS
  - 提供了空间数据类型、空间函数和空间索引
  - geom geometry (Point/Line/Polygon/Multixxx, 空间参考系)
  - ST_GeomFromText ('WKT表示', 空间参考系)
  - ST_XXX: ST_Distance, ST_Intersects, ST_DWithin, …
  - GiST

应用举例：打车软件

- 通常，数据库保留所有时间上的信息，即
  - Taxi(ID, driverID, ……, status, pos(Point, 4326), time)
  - User(ID, name, ……, pos(Point, 4326), time)
  - Road(ID, name, ……, line(LineString, 4326))
- 出租车(ID = B)附近1公里内的乘客叫车？
  - Select T.ID, T.position
    From Taxi T, User U
    Where T.ID = B and
    ST_Distance(T.pos, U.pos, true) < 1000
    and T.time ……
    and U.time ……
  - 这样的实现方式合理吗？

第九章 空间网络模型与查询

- 9.1 Motivation and use cases
- 9.2 Conceptual model
- 9.3 Logical model
  - 9.3.1 Transitive Closure
  - 9.3.2 CONNECT statement
  - 9.3.3 RECURSIVE statement
  - 9.3.4 RECURSIVE in PostgreSQL
- 9.4 Physical model
  - 9.4.1 Storage and data structures
  - 9.4.2 Algorithms for connectivity query and shortest path
- 9.5 pgRouting

参考教材：
Spatial Databases: A Tour Chapter 6
空间数据库管理系统概论，3.3-3.4

Navigation Systems

- Historical
  - Navigation is a core human activity for ages!
  - Trade-routes, Routes for Armed-Forces
- Recent Consumer Platforms
  - Devices: Phone Apps, In-vehicle, “GPS”, …
  - WWW: Google Maps, MapQuest, …
- Services
  - Display map around current location
  - Compute the shortest route to a destination
  - Help drivers follow selected route

Location Based Services

- Location: Where am I?
  - Geo-code: Place Name (or Street Address) → <latitude, longitude>
  - Reverse Geo-code: <latitude, longitude> → Place Name
- Directory: What is around me?
  - Where is the nearest Clinic? Restaurant? Taxi?
  - List all Banks within 1 mile
- Routes: How do I get there?
  - What is the shortest path to get there?
  - …

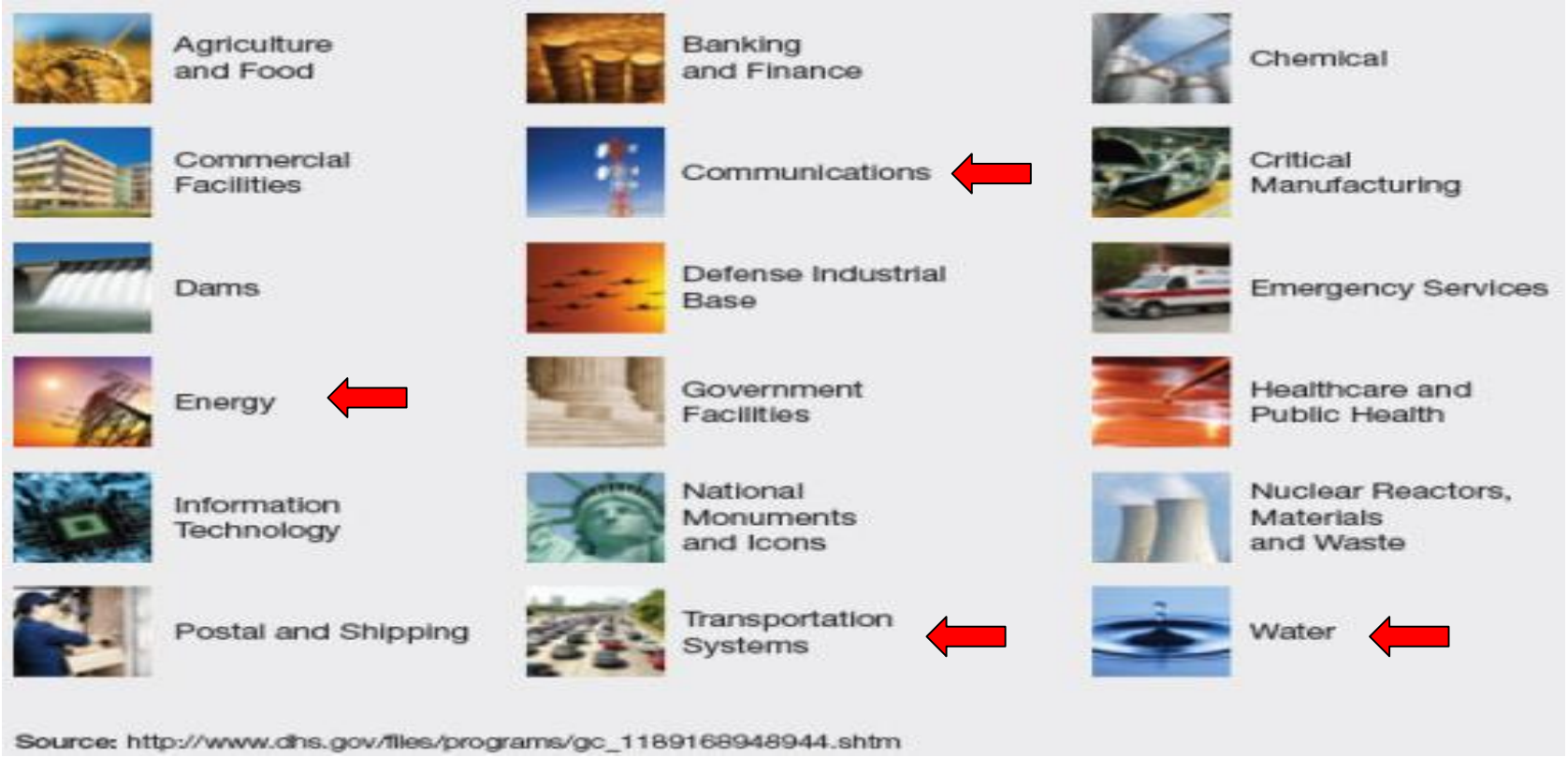
基于位置的服务

- 休闲娱乐型
  - 签到模式
  - 大富翁游戏模式
- 生活服务型
  - 周边生活服务的搜索
  - 与旅游的结合
  - 会员卡与票务模式
- 社交型
  - 地点交友，即时通讯
  - 以地理位置为基础的小型社区
- 商业型
  - LBS+团购；优惠信息推送服务；店内模式

http://blog.csdn.net/superjunjin/article/details/7818378

Spatial Networks & Modern Society

- Transportation, Energy, Water, Communications, …



Quiz 1

- Which of the following is not an application of spatial networks?
  - a) Navigation system
  - b) Geocoding
  - c) Reverse geocoding
  - d) Convex hull of a country

Limitations of Spatial Querying

- OGIS Simple Feature Types
  - Supports Geometry (e.g., Points, LineStrings, Polygons, …)
  - However, lack Graphs data type, shortest_path operator
- Traditional SQL
  - Supports select, project, join, statistics
  - Lacked transitive closure, e.g., network analysis (next slide)
  - SQL3 added recursion & transitive closure

思考：transitive closure与函数依赖的闭包之间的关系

Spatial Network Analysis

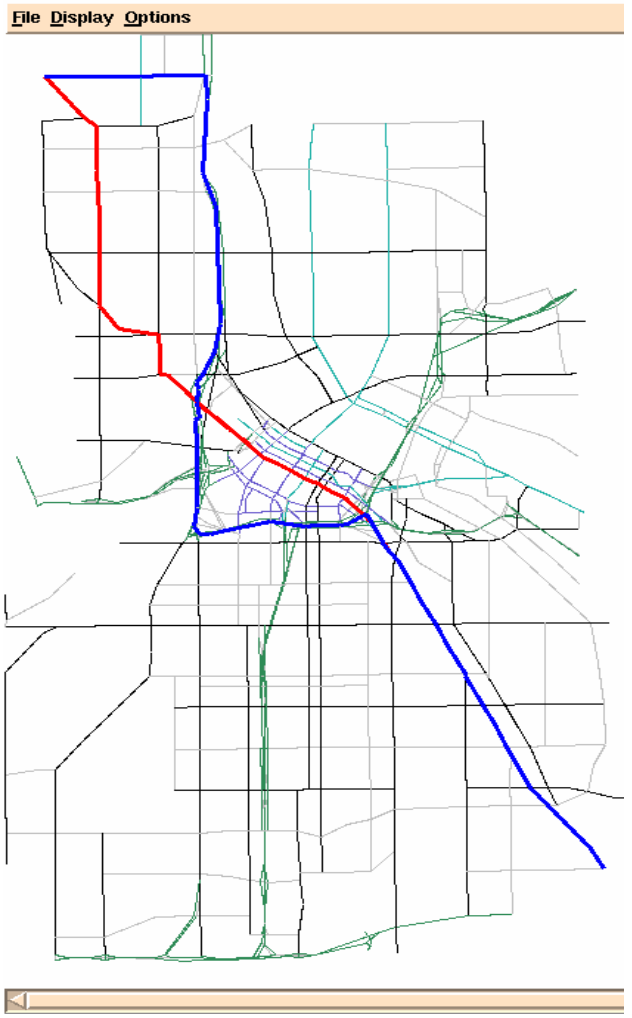
- Route (A start-point, Destination(s))
  - What is the shortest path to get there?
  - What is the shortest path to cover a set of destinations?
- Allocation (A set of service centers, A set of customers)
  - Assign customers to nearest service centers
  - Map service area for each service center
- Site Selection (A set of customers, Number of new service centers)
  - What are best locations for new service centers?

Quiz 2

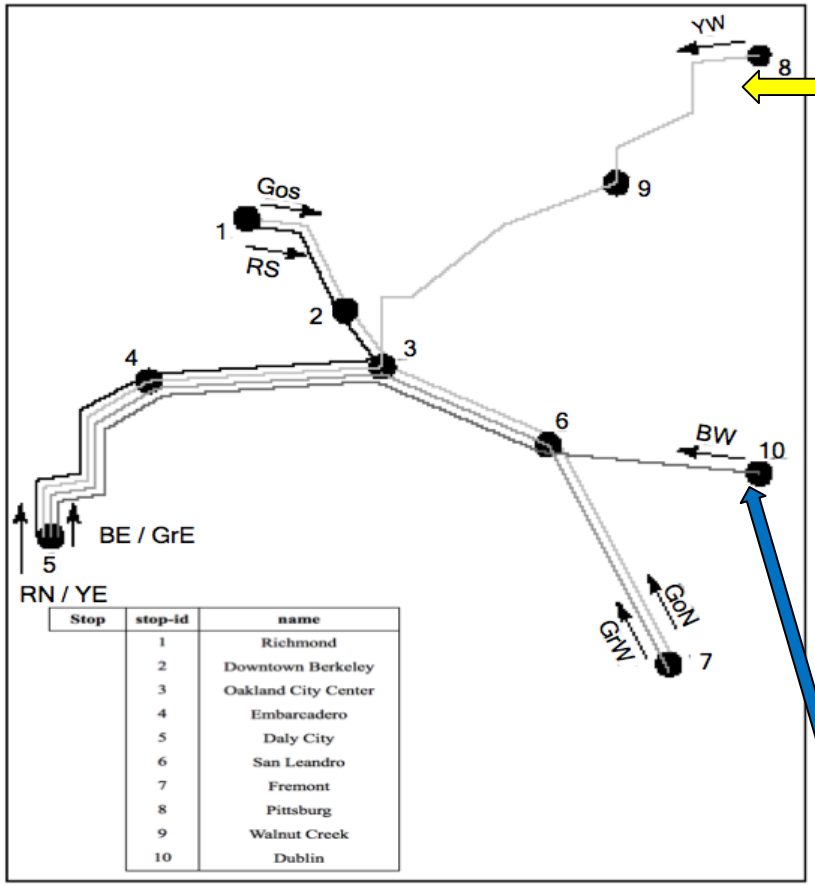
- Which of the following is not included in OGIS simple feature types?
  - a) Graph
  - b) Point
  - c) Line String
  - d) Polygon

Spatial Network Query Example

- Find shortest path from a start-point to a destination
- Find nearest hospital by driving distance
- Find shortest route to deliver packages to a set of homes
- Allocate customers to nearest service center



Railway Network & Queries



Stop	stop-id	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

- Find the number of stops on the Yellow West (YW) route
- List all stops which can be reached from Downtown Berkeley (2)
- List the routes numbers that connect Downtown Berkeley (2) & Daly City (5)
- Find the last stop on the Blue West (BW) route

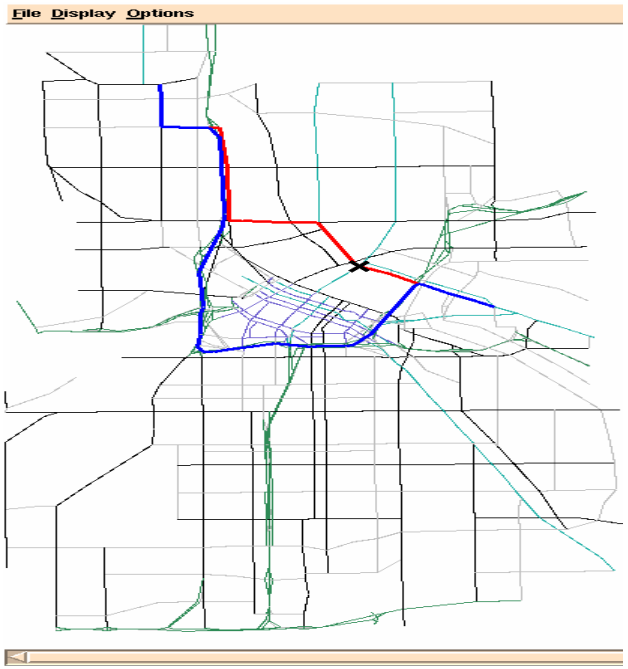
River Network & Queries

- List the names of all direct and indirect tributaries (支流) of Mississippi river
- List the direct tributaries of Colorado
- Which rivers could be affected if there is a spill in North Platte river

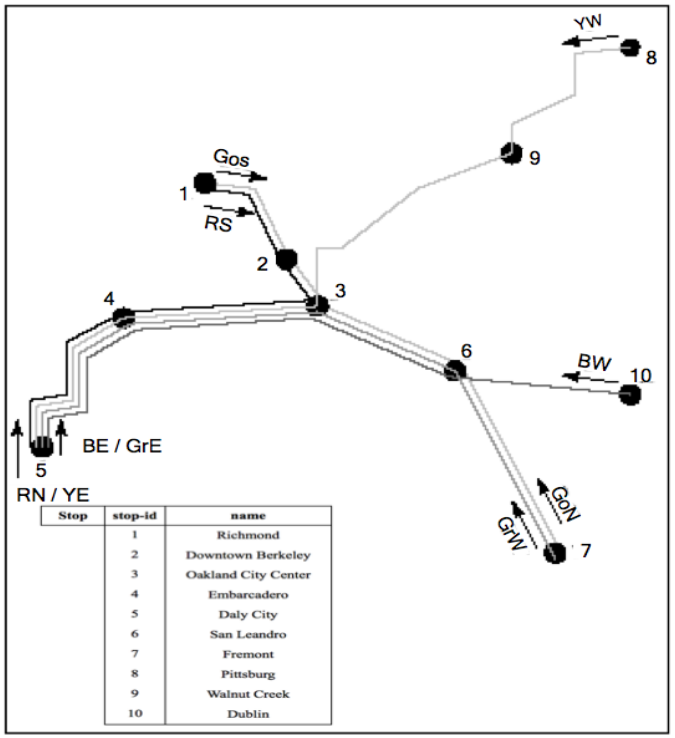
(b) River Network

Spatial Networks: Three Examples

A Road Network



A Railway Network



Stop	stop-id	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

A River Network

(b) River Network

Quiz 3

- Which of the following are queries on a road network?
  - a) What is the shortest path to the nearest airport?
  - b) What is the driving distance to the nearest gas station?
  - c) How many road intersections are there in my city?
  - d) All of the above

第九章 空间网络模型与查询

- 9.1 Motivation and use cases
- 9.2 Conceptual model
- 9.3 Logical model
  - 9.3.1 Transitive Closure
  - 9.3.2 CONNECT statement
  - 9.3.3 RECURSIVE statement
  - 9.3.4 RECURSIVE in PostgreSQL
- 9.4 Physical model
  - 9.4.1 Storage and data structures
  - 9.4.2 Algorithms for connectivity query and shortest path
- 9.5 pgRouting

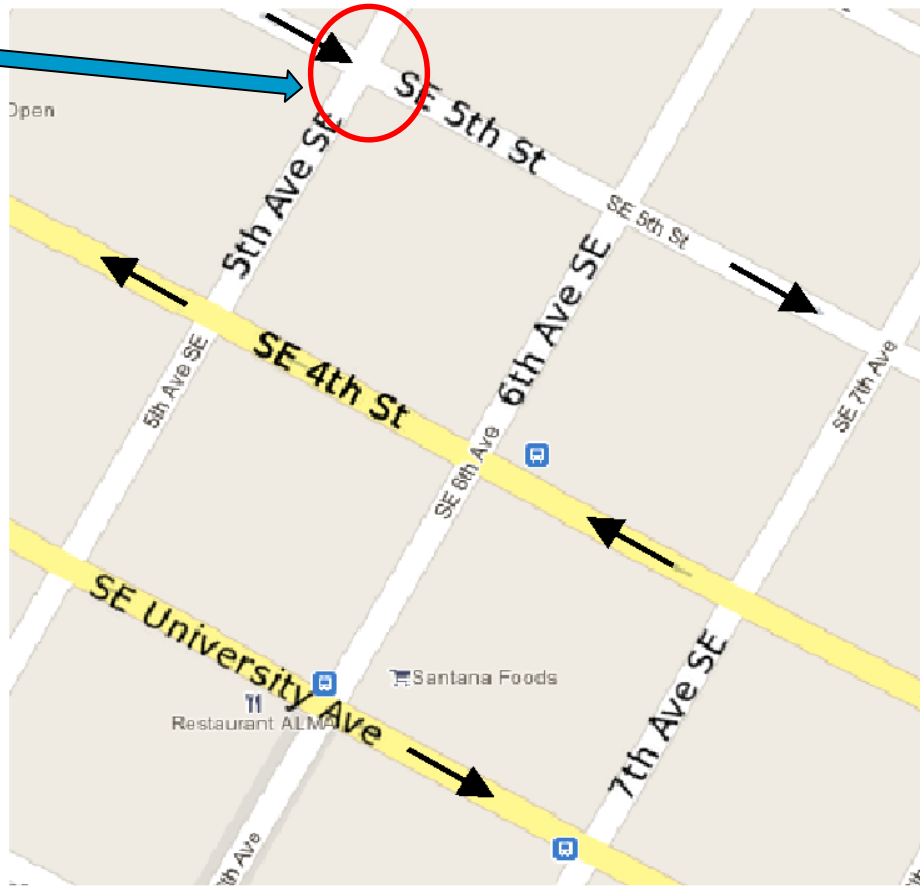
参考教材：
Spatial Databases: A Tour Chapter 6
空间数据库管理系统概论，3.3-3.4

Data Models of Spatial Networks

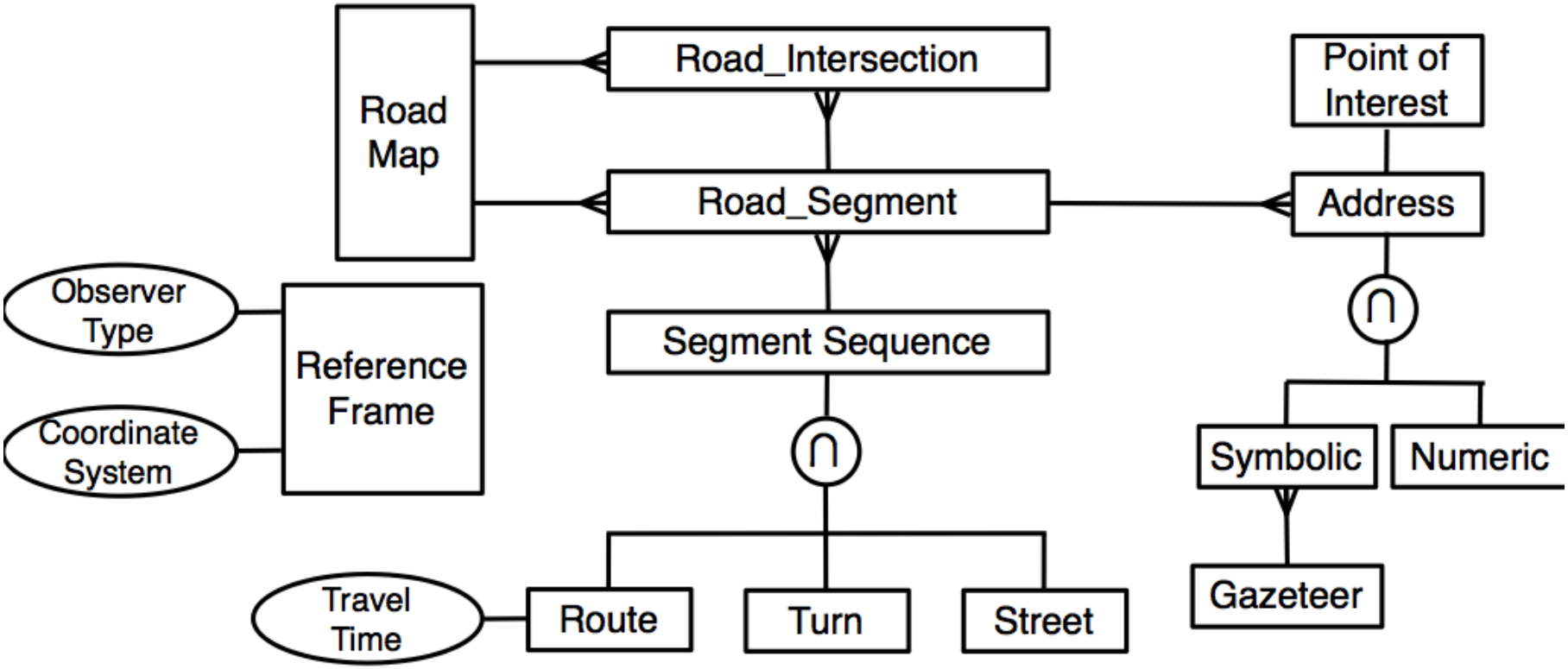
- Conceptual Model
  - Information Model: Entity Relationship Diagrams
  - Mathematical Model: Graphs
- Logical Data Model
  - Abstract Data types
  - Custom Statements in SQL
- Physical Data Model
  - Storage-Structures, File-Structures
  - Algorithms for common operations

Modeling Roadmaps

- Many Concepts, e.g.
  - Roads (or streets, avenues)
  - Road-Intersections
  - Road-Segments
  - Turns 
  - …

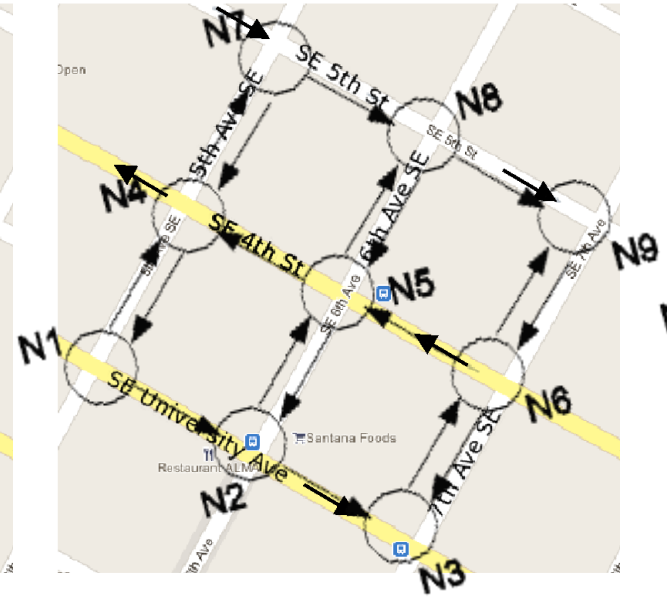


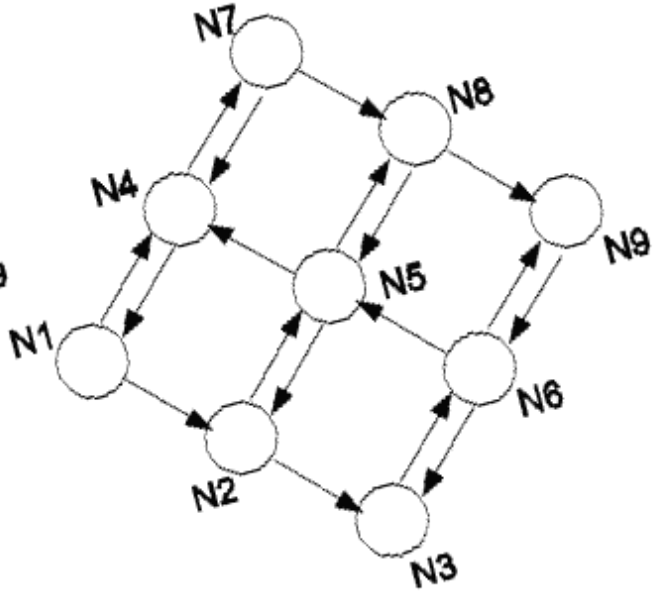
An Entity Relationship Diagram



Graph Models

- A Simple Mathematical Model
  - A graph G = (V,E)
  - V = a finite set of vertices
  - E = a set of edges model a binary relationship between vertices
- Example

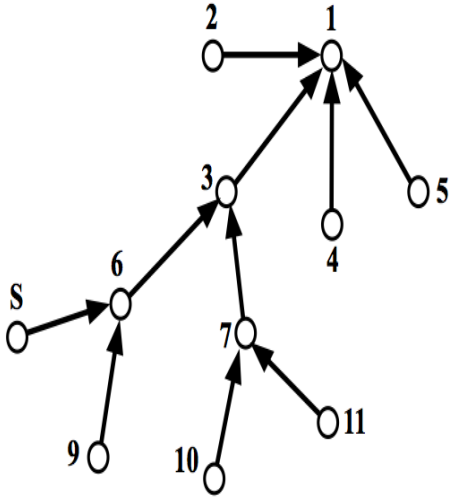


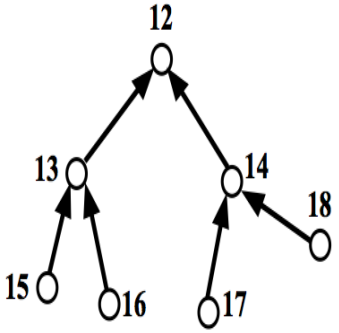


A Graph Model of River Network

- Nodes = rivers
- Edges = a river falls into another river

(b) River Network



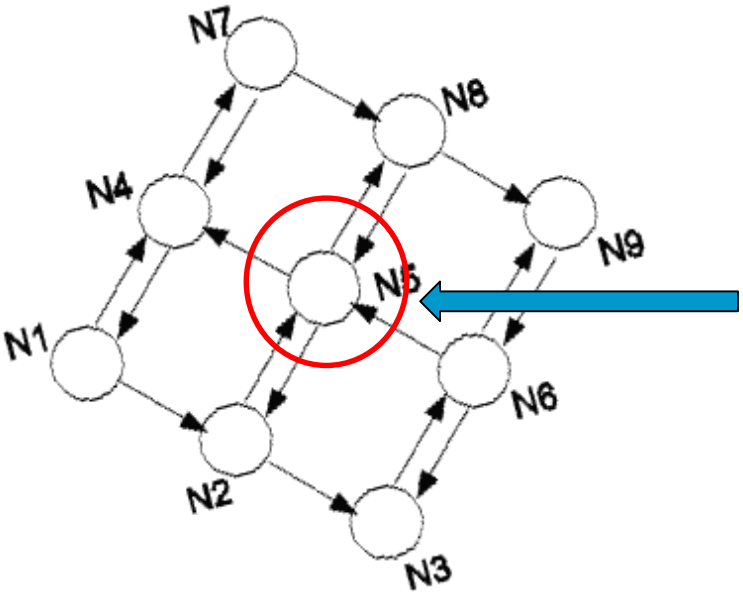


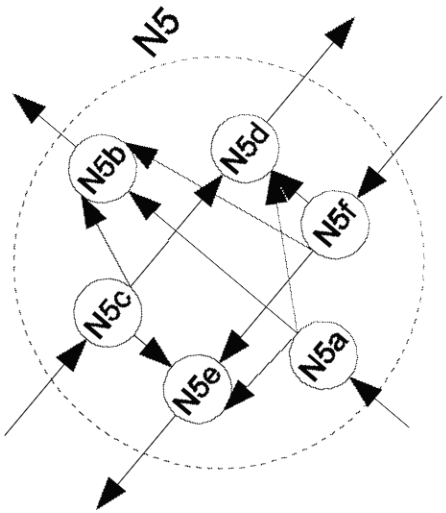
Pros and Cons of Graph Models

- Strength
  - Well developed mathematics for reasoning
  - Rich set of computational algorithms and data-structures
- Weakness
  - Models only one binary relationship
- Implications
  - A. Difficult to model multiple relationships, e.g., connect, turn
  - B. Multiple graph models possible for a spatial network

Modeling Turns in Roadmaps

- Approach 1: Model turns as a set of connects





- Approach 2: Use hyper-edges (and hyper-graphs)
- Approach 3: Annotate graph node with turn information

Alternative Graph Models for Roadmaps

- Choice 1:
  - Nodes = road-intersections
  - Edge (A, B) = road-segment connects adjacent road-intersections A, B
- Choice 2:
  - Nodes = (directed) road-segments
  - Edge (A, B) = turn from road-segment A to road-segment B
- Choice 3:
  - Nodes = roads
  - Edge(A, B) = road A intersects_with road B

Quiz 4

- Which of the following are usually not captured in common graph models of roadmaps?
  - a) Turn restrictions (e.g., no left turn)
  - b) Road intersections
  - c) Road segments
  - d) All of the above

第九章 空间网络模型与查询

- 9.1 Motivation and use cases
- 9.2 Conceptual model
- 9.3 Logical model
  - 9.3.1 Transitive Closure
  - 9.3.2 CONNECT statement
  - 9.3.3 RECURSIVE statement
  - 9.3.4 RECURSIVE in PostgreSQL
- 9.4 Physical model
  - 9.4.1 Storage and data structures
  - 9.4.2 Algorithms for connectivity query and shortest path
- 9.5 pgRouting

参考教材：
Spatial Databases: A Tour Chapter 6
空间数据库管理系统概论，3.3-3.4

Data Models of Spatial Networks

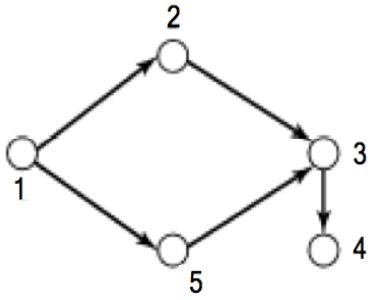
- Conceptual Model
  - Information Model: Entity Relationship Diagrams
  - Mathematical Model: Graphs
- Logical Data Model & Query Languages
  - Abstract Data types
  - Custom Statements in SQL
- Physical Data Model
  - Storage-Structures, File-Structures
  - Algorithms for common operations

9.3.1 Transitive Closure

- Consider a graph G = (V, E)
- Transitive closure(G) = G* = (V*, E*), where
  - V* = V
  - (A, B) in E* if and only if there is a path from A to B in G

Transitive Closure - Example

- 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

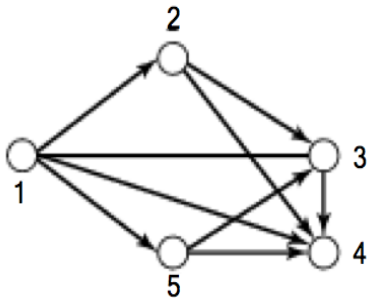


(a) Graph G

R

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

(b) Relation form



(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 relation form

Limitations of Original SQL

- Recall Relation Algebra based languages
  - Ex. original SQL
  - Can not compute transitive closure, e.g., shortest path

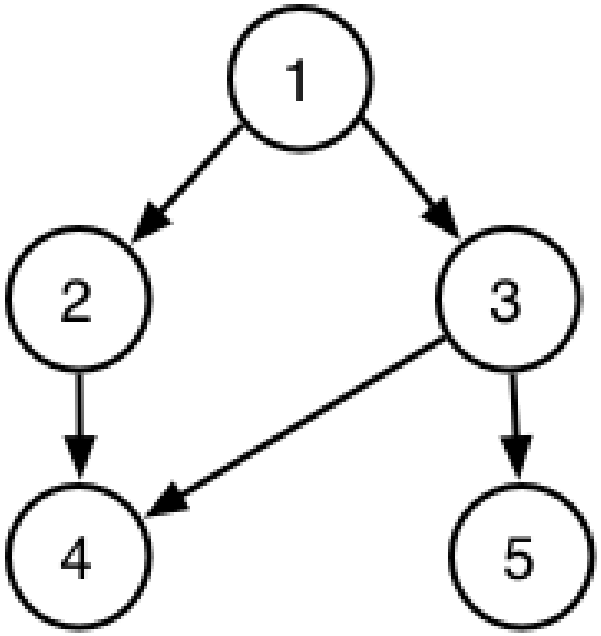
Supporting Graphs in SQL

- Abstract Data Type (user defined)
  - SQL3
  - May include shortest path operation!

- Custom Statements
  - SQL2 - CONNECT clause in SELECT statement
    - For directed acyclic graphs, e.g. hierarchies
  - SQL3 - WITH RECURSIVE statement
    - Transitive closure on general graphs
  - SQL3 - User defined data types
    - Can include shortest path operation!

Quiz 5

- Which of the following is not in the transitive closure of the following graph?
  - a) (1, 5)
  - b) (1, 4)
  - c) (2, 3)
  - d) (3, 4)



9.3.2 Querying Graphs: Overview

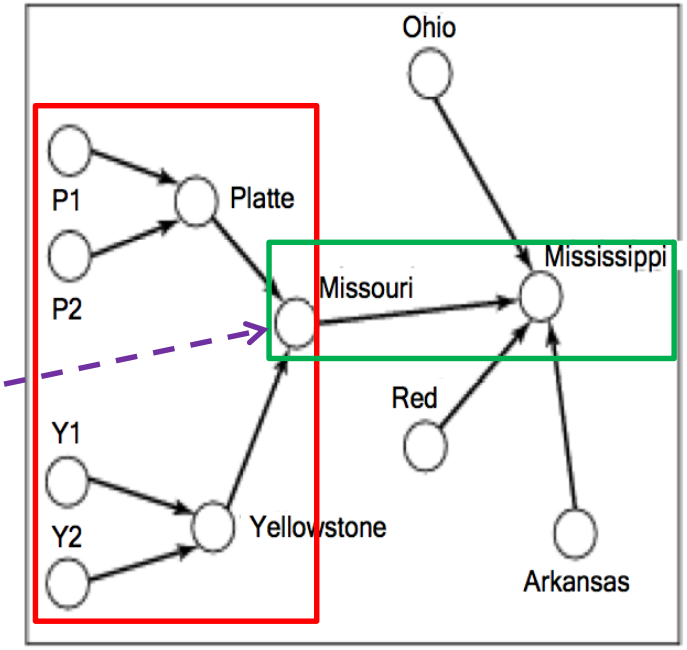
- Relational Algebra
  - Can not express transitive closure queries

- Two ways to extend SQL to support graphs
  - Abstract Data Types
  - Custom Statements
    - SQL2 - CONNECT BY clause(s) in SELECT statement
    - SQL3 - WITH RECURSIVE statement
    - SQL3 - User defined data types

CONNECT BY: Input, Output

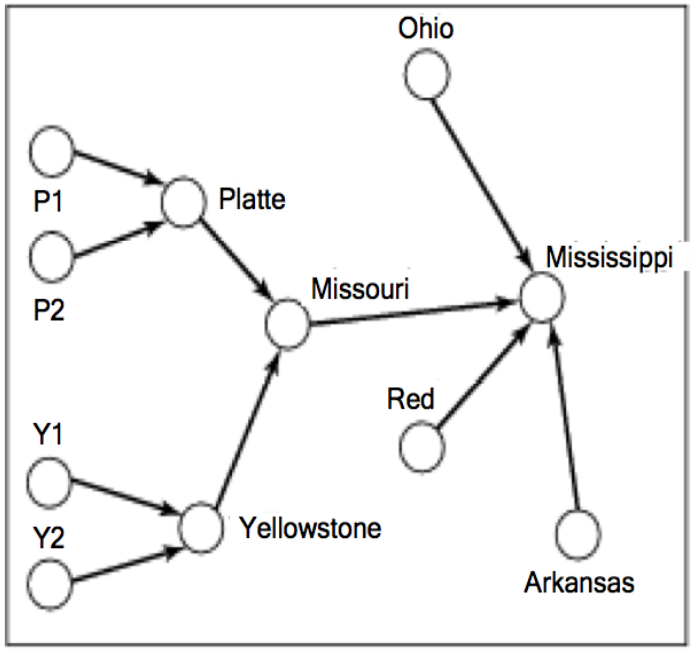
- Input:
  - (a) Edges of a directed acyclic graph G
  - (b) Start Node S, e.g., Missouri
  - (c) Travel Direction

- Output: Transitive closure of G
  - Ex. Predecessors of S = Missouri
  - Ex. Successors of S = Missouri (密苏里)



(a) Mississippi network (Y1 = Bighorn river, Y2 = Power river, P1 = Sweet water River, P2 = Big Thompson river)

Directed Edges: Tabular Representation



(a) Mississippi network (Y1 = Bighorn river, Y2 = Power river, P1 = Sweet water River, P2 = Big Thompson river)

Table: Falls_Into

Source	Dest
P1	Platte
P2	Platte
Y1	Yellowstone
Y2	Yellowstone
Platte	Missouri
Yellowstone	Missouri
Missouri	Mississippi
Ohio	Mississippi
Red	Mississippi
Arkansas	Mississippi

CONNECT BY– PRIOR - START WITH

```
SELECT  source
FROM      Falls_Into
CONNECT BY PRIOR  source = dest
START WITH dest = “Missouri”
```

Table: Falls_Into

Source	Dest
P1	Platte
P2	Platte
Y1	Yellowstone
Y2	Yellowstone
Platte	Missouri
Yellowstone	Missouri
Missouri	Mississippi
Ohio	Mississippi
Red	Mississippi
Arkansas	Mississippi

Q? What does CONNECT BY … PRIOR specify?

- Direction of travel
- Example: From Dest to Source
- Alternative: From Source to Dest

CONNECT BY– PRIOR - START WITH

- Syntax details
  - FROM clause a table for directed edges of an acyclic graph
  - PRIOR identifies direction of traversal for the edge
  - START WITH specifies first vertex for path computations
- Semantics
  - List all nodes reachable from first vertex using directed edge in specified table
  - Assumption - no cycle in the graph!
  - Not suitable for train networks, road networks

CONNECT BY– PRIOR - START WITH

Choice 1: Travel from Dest to Source

Ex. List direct & indirect tributaries of Missouri.

```
SELECT  source
FROM      Falls_Into
CONNECT BY PRIOR  source = dest
START WITH  dest =“Missouri”
```

Choice 2: Travel from Source to Dest

Ex. Which rivers are affected by spill in Missouri?

```
SELECT  dest
FROM      Falls_Into
CONNECT BY source = PRIOR dest
START WITH  source =“Missouri”
```

Table: Falls_Into

Source	Dest
P1	Platte
P2	Platte
Y1	Yellowstone
Y2	Yellowstone
Platte	Missouri
Yellowstone	Missouri
Missouri	Mississippi
Ohio	Mississippi
Red	Mississippi
Arkansas	Mississippi

Execution Trace – Step 1

```
SELECT  source
FROM      Falls_Into
CONNECT BY PRIOR  source = dest
START WITH dest = "Missouri"
```

1. Prior Result = SELECT * FROM Falls_Into
   WHERE dest = "Missouri"

Table: "Prior "

Source	Dest
Platte	Missouri
Yellowstone	Missouri

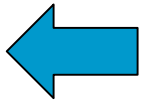


Table: Falls_Into

Source	Dest
P1	Platte
P2	Platte
Y1	Yellowstone
Y2	Yellowstone
Platte	Missouri
Yellowstone	Missouri
Missouri	Mississippi
Ohio	Mississippi
Red	Mississippi
Arkansas	Mississippi

Execution Trace – Step 2

```
SELECT  source
FROM      Falls_Into
CONNECT BY PRIOR  source = dest
START WITH  dest = “Missouri”
```

2. Iteratively add Join(Prior_Result.source = Falls_Into.dest)

```
Prior Result = SELECT * FROM Falls_Into
                    WHERE Prior_Result.source = Falls_Into.dest
```

Prior Result

Source
P1
P2
Y1
Y2
Platte
Yellowstone

Prior Result

Source	Dest
Platte	Missouri
Yellowstone	Missouri

Table: Falls_Into

Source	Dest
P1	Platte
P2	Platte
Y1	Yellowstone
Y2	Yellowstone
Platte	Missouri
Yellowstone	Missouri
Missouri	Mississippi
Ohio	Mississippi
Red	Mississippi
Arkansas	Mississippi

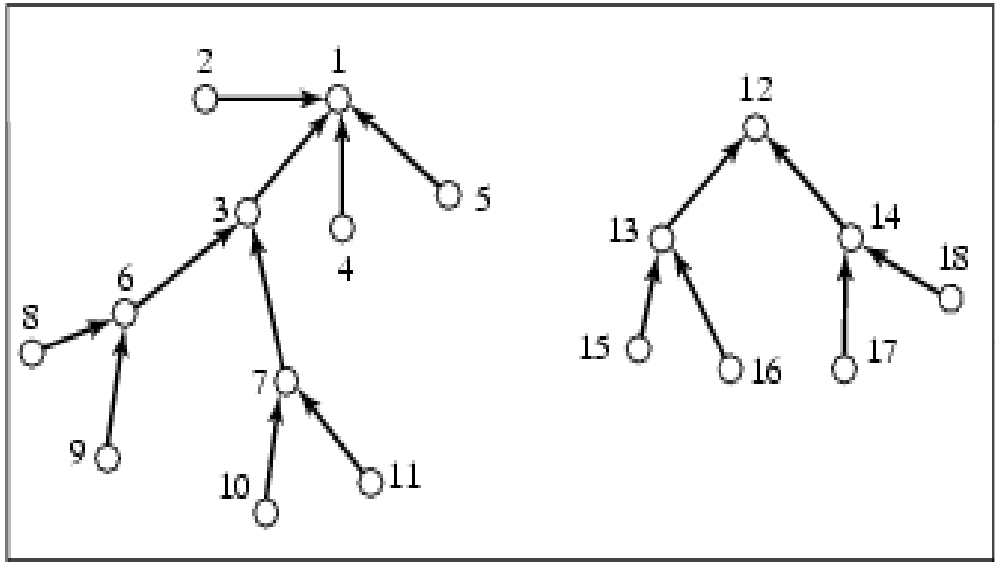
SQL CONNECT Exercise

- Study 2 SQL queries on right
  - Note different use of PRIOR keyword

- Compute results of each query
- Which one returns ancestors of 3?
- Which returns descendents of 3?
- Which query lists river affected by oil spill in Missouri (id = 3)?

SELECT source **FROM** FallsInto
CONNECT BY PRIOR source = dest
START WITH dest = 3

SELECT dest **FROM** FallsInto
CONNECT BY source = **PRIOR** dest
START WITH source = 3



Quiz 6

- Which of the following is false about CONNECT BY clause?
  - a) It is only able to output predecessors, but not successors, of the start node
  - b) It is able to output transitive closure of a directed graph
  - c) It usually works with PRIOR and START WITH keywords
  - d) None of the above

9.3.3 Querying Graphs: Overview

- Relational Algebra
  - Can not express transitive closure queries

- Two ways to extend SQL to support graphs
  - Abstract Data Types
  - Custom Statements
    - SQL2 - CONNECT BY clause(s) in SELECT statement
    - SQL3 - WITH RECURSIVE statement
    - SQL3 - User defined data types

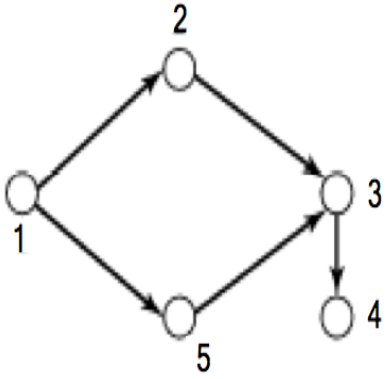
WITH RECURSIVE: Input, Output

- Input:
  - (a) Edges of a directed graph G
  - (b) Sub-queries to
    - Initialize results
    - Recursively grow results
    - Additional constraints

R

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

(b) Relation form



(a) Graph G

- Output: Transitive closure of G
  - Ex. Predecessors of a node
  - Ex. Successors of a node

Syntax of WITH RECURSIVE Statement

```
WITH RECURSIVE X(source, dest)
AS (SELECT source, dest FROM R )
        UNION
(SELECT R.source,  X.dest
        FROM R,  X
        WHERE R.dest = X.source )
```

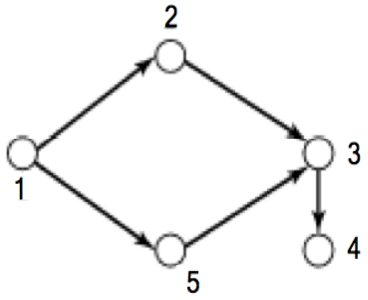
Description of Result Table

Initialization Query

Recursive Query to grow result

Example Input and Output

```
WITH RECURSIVE  X(source,dest)
AS  (SELECT source,dest FROM R )
            UNION
      (SELECT R.source,  X.dest
            FROM R,  X
            WHERE R.dest=X.source )
```

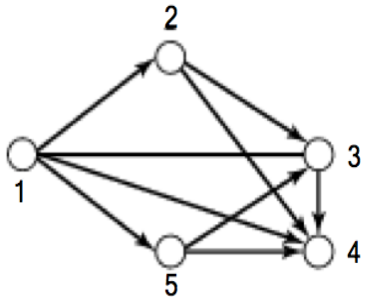


(a) Graph G

R

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

(b) Relation form



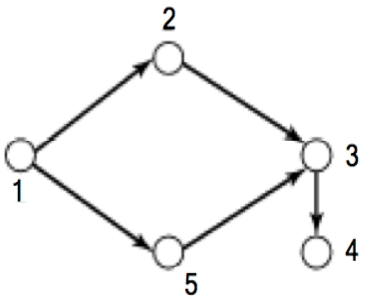
(c) Transitive closure (G) = Graph G

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 relation form

SQL3 Recursion Example - Meaning

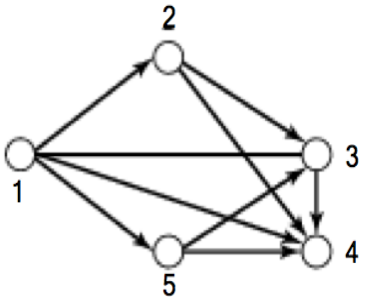


(a) Graph G

R

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

(b) Relation form



(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 relation form

- Initialize X by
  (**SELECT** source,dest **FROM** R)
- Recursively grow X by
  (**SELECT** R.source, X.dest
  FROM R, X
  WHERE R.dest=X.source)
- Infer X(a,c) from R(a,b),X(b,c)
- Infer X(1,3) from R(1,2),X(2,3)
- Infer X(2,4) from R(2,3),X(3,4)
- Infer X(5,4) from R(5,3),X(3,4)
- Infer X(1,4) from R(1,5),X(5,4)

SQL3 Recursion

- Syntax
  - WITH RECURSIVE <Relational Schema>
  - AS <Query to populate relational schema> Syntax details
  - <Relational Schema> lists columns in result table with directed edges
  - <Query to populate relational schema> has UNION of nested sub-queries
    - Base cases to initialize result table
    - Recursive cases to expand result table

SQL3 Recursion

- Semantics
  - Results relational schema say X(source, dest)
    - Columns source and dest come from same domain, e.g. Vertices
    - X is a edge table, X(a,b) directed from a to b
  - Result table X is initialized using base case queries
  - Result expanded using X(a, b) and X(b, c) implies X(a, c)

With Recursive for Connectivity

- Connect by
  - For directed acyclic graphs, e.g. hierarchies
  - PostgreSQL, SQL Server not supported
  - Oracle supported
- With Recursive
  - Transitive closure on general graphs
  - PostgreSQL, SQL Server, Oracle supported

http://www.postgresql.org/docs/current/static/queries-with.html

Case Studies

- Goal: Compare relational schemas for spatial networks
  - River networks has an edge table, Falls_Into
  - BART train network does not an edge table
  - Edge table is crucial for using SQL transitive closure
  - Exercise: Proposed a different set of table to model BART as a graph
    - using an edge table connecting stops
- River networks - graph model
  - Can use SQL transitive closure to compute ancestors or descendent of a river
  - We saw an examples using CONNECT BY clause
  - Exercises explore use of WITH RECURSIVE statement

Case Studies

- BART train network - non-graph model
  - Entities = Stop, Route
  - Relationship = aMemberOf(Stop, Route)
  - Can not use SQL recursion
    - No table can be viewed as edge table
    - RouteStop table is a subset of transitive closure

- Transitive closure queries on edge(from_stop, to_stop)
  - A few can be answered by querying RouteStop table
  - Many can not be answered
    - Find all stops reachable from Downtown Berkeley

Quiz 7

- Which of the following are true about WITH RECURSIVE clause?
  - a) It is able to output transitive closure of a directed graph
  - b) It usually works with an edge table
  - c) It includes two SELECT statements
  - d) All of the above

9.3.4 With Recursive in PostgreSQL

- With clause
  - A way to write auxiliary statements for use in a larger query
  - Define temporary tables that exist just for one query
- A WITH query can refer to its own output
- Sum the integers from 1 through 100

```
WITH RECURSIVE t(n) AS (
        VALUES (1)
    UNION ALL
        SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t;
```

A non-recursive term

Union or Union ALL

A recursive term

http://www.postgresql.org/docs/current/static/queries-with.html

Recursive Query Evaluation

- Step 1: Evaluate the non-recursive term
  - For UNION, discard duplicate rows
  - Include all remaining rows in the result of the recursive query, and also place them in a temporary working table
  - Example
    - VALUES (1)

```
WITH RECURSIVE t(n) AS (
    VALUES (1)
  UNION ALL
    SELECT n+1 FROM t WHERE n < 100)
```

Query Result	Working
1	1

Recursive Query Evaluation

- Step 2: So long as working table is not empty, repeat these steps:
  - 2.1 Evaluate the recursive term, substituting the current contents of the working table for the recursive self-reference
  - Example
    - SELECT n+1 FROM t WHERE n < 100
    - t is the query result table Q?
      or the working table W

```
WITH RECURSIVE t(n) AS (
    VALUES (1)
  UNION ALL
    SELECT n+1 FROM t WHERE n < 100)
```

Intermediate	Query Q	Working W
2	1	1

Recursive Query Evaluation

- Step 2: So long as working table is not empty, repeat these steps:
  - 2.1 Evaluate the recursive term, substituting the current contents of the working table for the recursive self-reference
  - 2.2 For UNION, discard duplicated rows and rows that duplicated any previous result row
    - Where are previous result rows?

```
WITH RECURSIVE t(n) AS (
    VALUES (1)
  UNION ALL
    SELECT n+1 FROM t WHERE n < 100)
```

Intermediate	Query Q	Working W
2	1	1

Recursive Query Evaluation

- Step 2: So long as working table is not empty, repeat these steps:
  - 2.1 Evaluate the recursive term, substituting the current contents of the working table for the recursive self-reference
  - 2.2 For UNION, discard duplicated rows and rows that duplicated any previous result row
  - 2.3 Including all remaining rows in the result of the recursive query, and also place them in a temporary intermediate table

Intermediate	Query Q	Working W
2	1	1
	2	

Recursive Query Evaluation

- Step 2: So long as working table is not empty, repeat these steps:
  - 2.1 Evaluate the recursive term, substituting the current contents of the working table for the recursive self-reference
  - 2.2 For UNION, discard duplicated rows and rows that duplicated any previous result row
  - 2.3 Including all remaining rows in the result of the recursive query, and also place them in a temporary intermediate table
  - 2.4 Replace the contents of the working table with the contents of the intermediate table, then empty the intermediate table

Recursive Query Evaluation

- Step 2: So long as working table is not empty, repeat these steps
  - SELECT n+1 FROM W WHERE n < 100

Intermediate	Query Q	Working W
	1	2
	2	

Intermediate	Query Q	Working W
3	1	2
	2	

Intermediate	Query Q	Working W
	1	3
	2	
	3	

Recursive Query Evaluation

- Step 2: So long as working table is not empty, repeat these steps
  - SELECT n+1 FROM W WHERE n < 100

```
WITH RECURSIVE t(n) AS (
    VALUES (1)
  UNION ALL
    SELECT n+1 FROM t WHERE n < 100)
```

t
1
……
100

Intermediate	Query Q	Working W
	1	100
	……	
	100	

Intermediate	Query Q	Working W
	1	
	……	
	100	

Recursive Query Evaluation in C

- with recursive X(…) as (
  SQL_A(O)
  union / union all
  SQL_B(O, X))

O：原始关系/表
X：最终关系/表
W：临时关系/表
T：临时关系/表

- X(…) = SQL_A[remove duplicates for union];
  W(…) = X(…);
  while (table W is not empty) {
    T(…) = SQL_B(O, W) [except X(…) for union] ;
    X(…) = X(…) union / union all T(…);
    W(…) = T(…);
  }

问题：Union与Union ALL区别
避免使用Union ALL

Recursive Query Evaluation in C

```
WITH RECURSIVE t(n) AS (
        VALUES (1)
    UNION ALL
        SELECT n+1 FROM t WHERE n < 100 )
```

t：最终关系/表
W：临时关系/表
T：临时关系/表

- insert into t(n) values(1);

```
delete from W; insert into W(n) select * from t;
while ((select count(*) from W) <> 0) {
    delete from T;
    insert into T(n) select n+1 from W where n < 100;
    insert into t(n) select * from T; // union all
    delete from W; insert into W from select * from T;
}
```

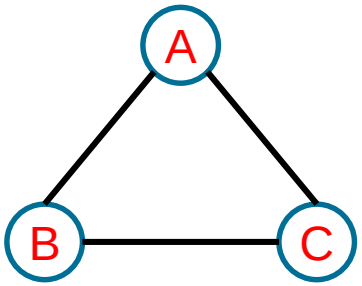
With Recursive Limitation

- Strictly speaking, this process is iteration not recursion, but RECURSIVE is the terminology chosen by the SQL standards committee
- Recursive queries are typically used to deal with hierarchical or tree-structured data

With Recursive Limitation

- Important: the recursive part of the query will eventually return no tuples, or else the query will loop indefinitely
  - Using UNION instead of UNION ALL can accomplish this by discarding rows that duplicate previous output rows
- A cycle does not involve output rows that are completely duplicate
  - It may be necessary to check just one or a few fields to see if the same point has been reached before
- Standard method
  - Compute an array of the already-visited values

With Recursive Example 1



- 广度优先遍历 - 深度depth
- 数据库中，右图的关系为edges(start, end)
  - 6行记录(A,B), (A,C), (B,A), (B,C), (C,A), (C,B)
- with recursive X(node, depth) as (

  select start, 0 from edges where start = A

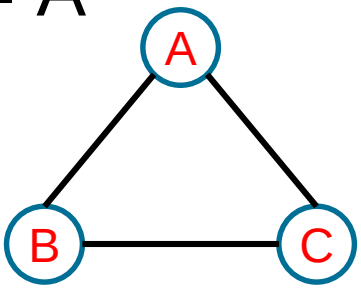
  union

  select end, depth + 1 from edges, X

  where start = node and depth < 3)
- 如果不加depth<3，上述语句会死循环

With Recursive Example 1

- with recursive X(node, depth) as (
  select start, 0 from edges where start = A
  union
  select end, depth + 1 from edges, X
  where start = node and depth < 3)



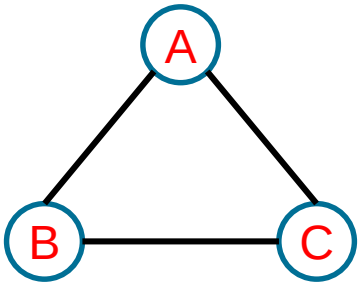
- 初始X：(A, 0)
- 第1次迭代：(A, 0), (B, 1), (C, 1)
- 第2次迭代：(A, 0), (B, 1), (C, 1), (A, 2), (B, 2), (C, 2)
  - 如果使用union all得到: (A, 2), (C, 2), (A, 2), (B, 2)
- 第3次迭代：(A, 0), (B, 1), (C, 1), (A, 2), (B, 2), (C, 2) (A, 3), (B, 3), (C, 3)
- ……

With Recursive Example 2

- 避免重复遍历节点path和cycle
- with recursive X(node, depth, path, cycle) as (
  select start, 0, array[start], false from edges where start = A
  union
  select end, depth + 1, path || end, end = any(path)
  from edges, X
  where start = node and not cycle)
- PostgreSQL中的数组
  - 初始化：array[…]
  - 增加元素：path || end
  - 判断元素是否在数组中：end = any(path)

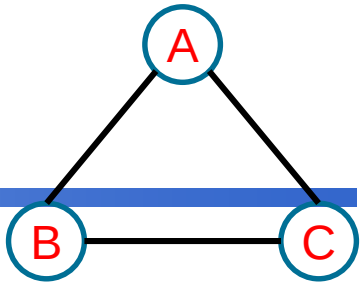
With Recursive Example 2

- with recursive X(node, depth, path, cycle) as (
  select start, 0, array[start], false from edges where start = A
  union
  select end, depth + 1, path || end, end = any(path)
  from edges, X
  where start = node and not cycle)



- 初始X：(A, 0, [A], false)
- 第1次迭代：(A, 0, [A], false), (B, 1, [A, B], false), (C, 1, [A, C], false)
- 第2次迭代：(A, 0, [A], false), (B, 1, [A, B], false), (C, 1, [A, C], false),
  (A, 2, [A, B, A], true), (C, 2, [A, B, C], false),
  (A, 2, [A, C, A], true), (B, 2, [A, C, B], false)
- 第3次迭代：(A, 0, [A], false), (B, 1, [A, B], false), (C, 1, [A, C], false),
  (A, 2, [A, B, A], true), (C, 2, [A, B, C], false),
  (A, 2, [A, C, A], true), (B, 2, [A, C, B], false),
  (A, 3, [A, B, C, A], true), (B, 3, [A, B, C, B], true),
  (A, 3, [A, C, B, A], true), (C, 3, [A, C, B, C], true)

With Recursive Example 2



- 初始X：(A, 0, [A], false)
- 第1次迭代：(A, 0, [A], false), (B, 1, [A, B], false), (C, 1, [A, C], false)
- 第2次迭代：(A, 0, [A], false), (B, 1, [A, B], false), (C, 1, [A, C], false),
  (A, 2, [A, B, A], true), (C, 2, [A, B, C], false),
  (A, 2, [A, C, A], true), (B, 2, [A, C, B], false)
- 第3次迭代：(A, 0, [A], false), (B, 1, [A, B], false), (C, 1, [A, C], false),
  (A, 2, [A, B, A], true), (C, 2, [A, B, C], false),
  (A, 2, [A, C, A], true), (B, 2, [A, C, B], false),
  (A, 3, [A, B, C, A], true), (B, 3, [A, B, C, B], true),
  (A, 3, [A, C, B, A], true), (C, 3, [A, C, B, C], true)
- 第4次迭代：(A, 0, [A], false), (B, 1, [A, B], false), (C, 1, [A, C], false),
  (A, 2, [A, B, A], true), (C, 2, [A, B, C], false),
  (A, 2, [A, C, A], true), (B, 2, [A, C, B], false),
  (A, 3, [A, B, C, A], true), (B, 3, [A, B, C, B], true),
  (A, 3, [A, C, B, A], true), (C, 3, [A, C, B, C], true)
- 结束

With Recursive Example 3

- Find all the direct and indirect sub-parts of a product, given only a table that shows immediate inclusions
  - parts(part, sub_part, quantity)
  - Similar to Rivers

With Recursive Example 3

- Find all the direct and indirect sub-parts of a product

```
WITH RECURSIVE included_parts(sub_part, part, quantity) AS (
SELECT sub_part, part, quantity FROM parts WHERE part = 'our_product'
UNION ALL
SELECT p.sub_part, p.part, p.quantity FROM included_parts pr, parts p
WHERE p.part = pr.sub_part
)

SELECT sub_part, SUM(quantity) as total_quantity
FROM included_parts
GROUP BY sub_part
```

With Recursive Example 4

- Graph search: graph(id, link, data)

```
WITH RECURSIVE search_graph(id, link, data, depth) AS (
        SELECT g.id, g.link, g.data, 1
        FROM graph g
      UNION ALL
        SELECT sg.id, g.link, g.data, sg.depth + 1
        FROM graph g, search_graph sg
        WHERE g.id = sg.link
)
SELECT * FROM search_graph;
```

- This query will loop if the link relationship contain cycle
  - UNION ALL → UNION
    - Would not eliminate the looping due to the “depth” output

With Recursive Example 4

- Need to recognize whether we have reached the same row again while following path of links
  - Add two columns path and cycle to the loop-prone query

```
WITH RECURSIVE search_graph(id, link, data, depth, path, cycle) AS (
        SELECT g.id, g.link, g.data, 1,
               ARRAY[g.id],
               false
        FROM graph g
    UNION ALL
        SELECT sg.id, g.link, g.data, sg.depth + 1,
               path || g.id,
               g.id = ANY(path)
        FROM graph g, search_graph sg
        WHERE g.id = sg.link AND NOT cycle
)
```

Represent the “path” taken to reach any particular row

With Recursive Example 4

- Generally, more than one field needs to be checked to recognize a cycle, use an array of rows

```
WITH RECURSIVE search_graph(id, link, data, depth, path, cycle) AS (
        SELECT g.id, g.link, g.data, 1,
               ARRAY[ROW(g.f1, g.f2)],
               false
        FROM graph g
    UNION ALL
        SELECT g.id, g.link, g.data, sg.depth + 1,
               path || ROW(g.f1, g.f2),
               ROW(g.f1, g.f2) = ANY(path)
        FROM graph g, search_graph sg
        WHERE g.id = sg.link AND NOT cycle
)
SELECT * FROM search_graph;
```

Recursive in PostgreSQL

- A helpful trick for testing queries when you are not certain if they might loop is to place a LIMIT in the parent query

```
WITH RECURSIVE t(n) AS (
        VALUES (1)
    UNION ALL
        SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t LIMIT 100;
```

避免死循环方法

1. Union
2. Limit
3. Depth控制
4. Cycle检测

- PostgreSQL evaluates only as many rows of a WITH query as are actually fetched by the parent query
  - Other DBMS might work different
  - Won’t work if order by or join them to some other table

With Queries Applications

- Avoid redundant work
  - Evaluated only once per execution of the parent query, even if they are referred to more than once by the parent query or sibling WITH queries
  - Expensive calculations that are needed in multiple places can be placed within a WITH query
- Prevent unwanted multiple evaluations of functions with side-effects
  - The optimizer is less able to push restrictions from the parent query down into a WITH query than an ordinary sub-query

第九章 空间网络模型与查询

- 9.1 Motivation and use cases
- 9.2 Conceptual model
- 9.3 Logical model
  - 9.3.1 Transitive Closure
  - 9.3.2 CONNECT statement
  - 9.3.3 RECURSIVE statement
  - 9.3.4 RECURSIVE in PostgreSQL
- 9.4 Physical model
  - 9.4.1 Storage and data structures
  - 9.4.2 Algorithms for connectivity query and shortest path
- 9.5 pgRouting

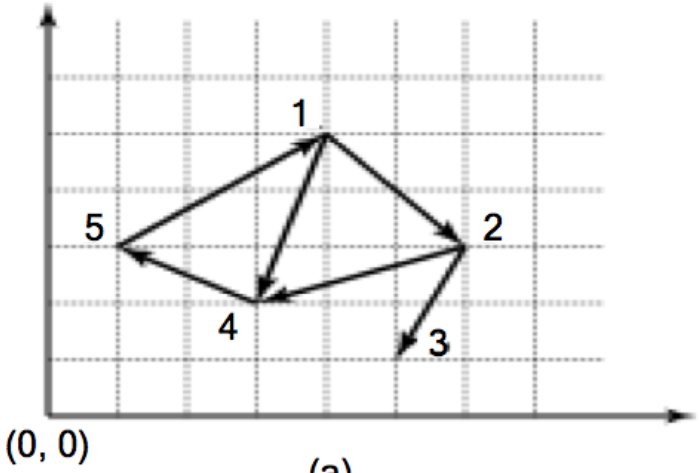
参考教材：
Spatial Databases: A Tour Chapter 6
空间数据库管理系统概论，3.3-3.4

Data Models of Spatial Networks

- Conceptual Model
  - Entity Relationship Diagrams, Graphs
- Logical Data Model & Query Languages
  - Abstract Data types
  - Custom Statements in SQL
- Physical Data Model
  - Storage: Data-Structures, File-Structures
  - Algorithms for common operations

9.4.1 Main Memory Data-Structures

- Adjacency matrix
  - M[A, B] = 1 if and only if edge(vertex A, vertex B) exists
- Adjacency list
  - Maps a vertex to a list of its successors

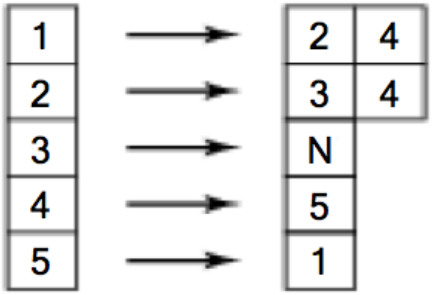


(a)

Destination

source	1	2	3	4	5
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

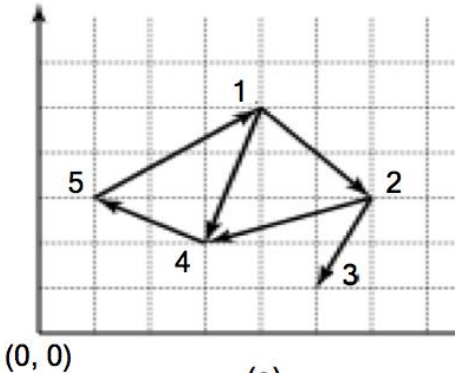
(b) Adjacency-Matrix



(c) Adjacency-List

Disk-based Tables

- Normalized tables
  - One for vertices, other for edges
- Denormalized
  - One table for nodes with adjacency lists



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

Spatial Network Storage

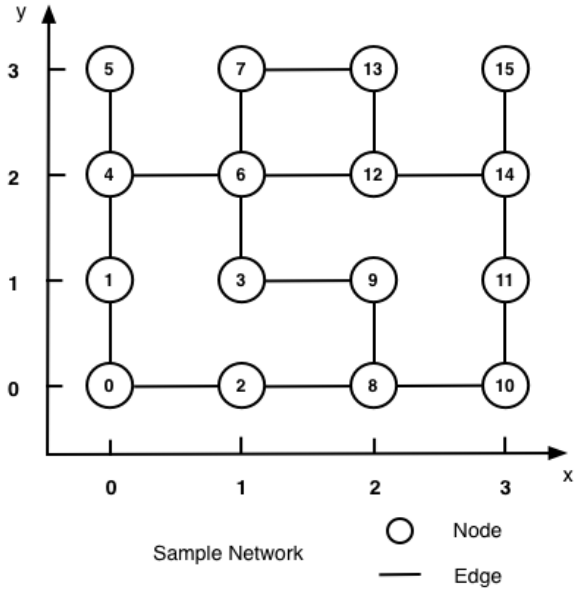
- Problem Statement
  - Given a spatial network
  - Find efficient data-structure to store it on disk sectors
  - Goal - Minimize I/O-cost of operations
    - Find(), Insert(), Delete(), Create()
    - Get-A-Successor(), Get-Successors()
  - Constraints
    - Spatial networks are much larger than main memories
- Problems with geometric indices, e.g. R-tree
  - Clusters objects by proximity not edge connectivity
  - Performs poorly if edge connectivity not correlated with proximity (邻近)
- Trends: graph based methods

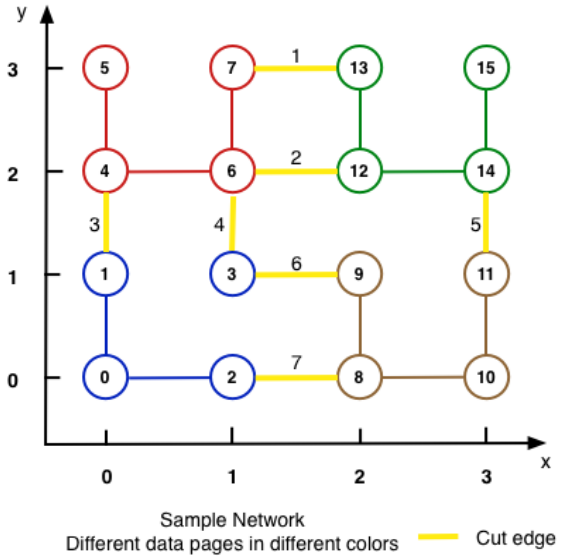
Graph Based Storage Methods

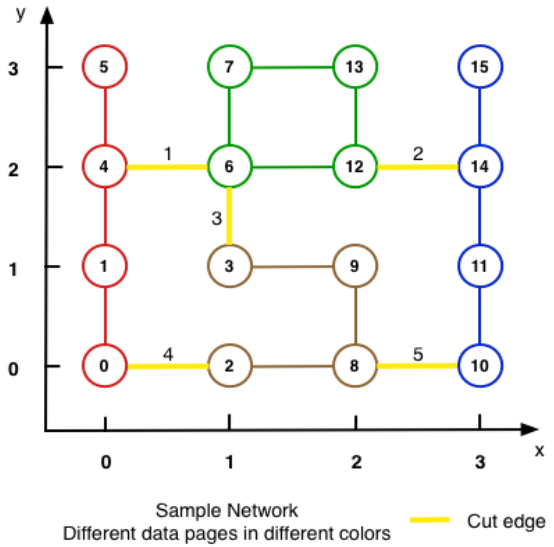
- Insights
  - I/O cost of operations (e.g. get-a-successor) minimized by maximizing CRR
  - CRR = Pr. (node-pairs connected by an edge are together in a disk sector)

File-Structures:
Partition Graph into Disk Blocks

- Which partitioning reduces disk I/O for graph operations?
  - Choice 1: Geometric partition
  - Choice 2: Min-cut Graph Partition
  - Choice 2 cuts fewer edges and is preferred
  - Assuming uniform querying popularity across edges

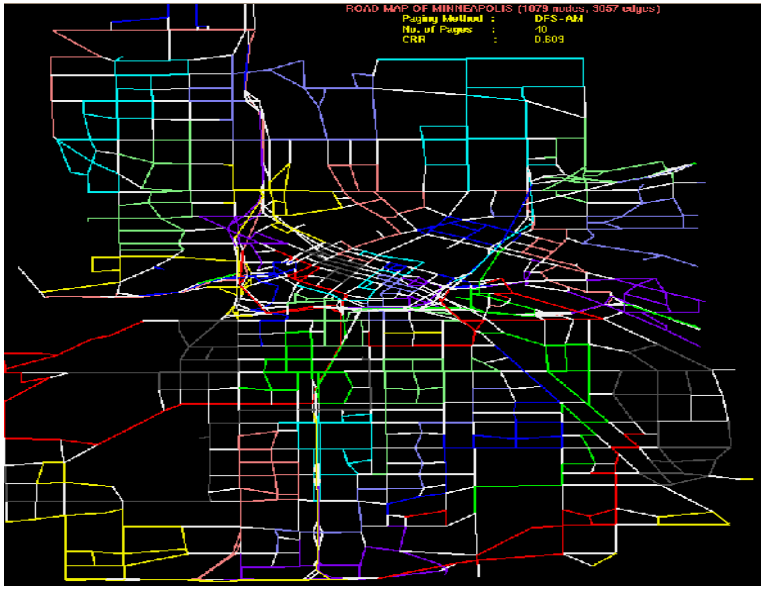


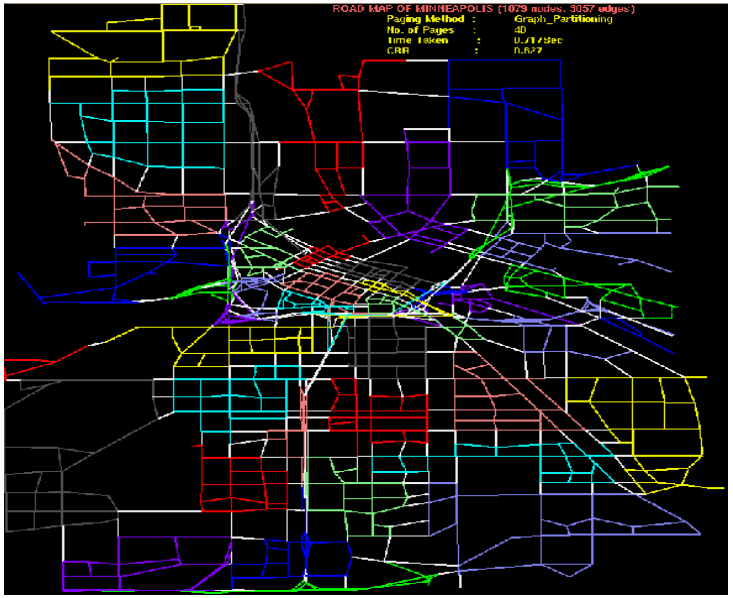




Graph Based Storage Methods

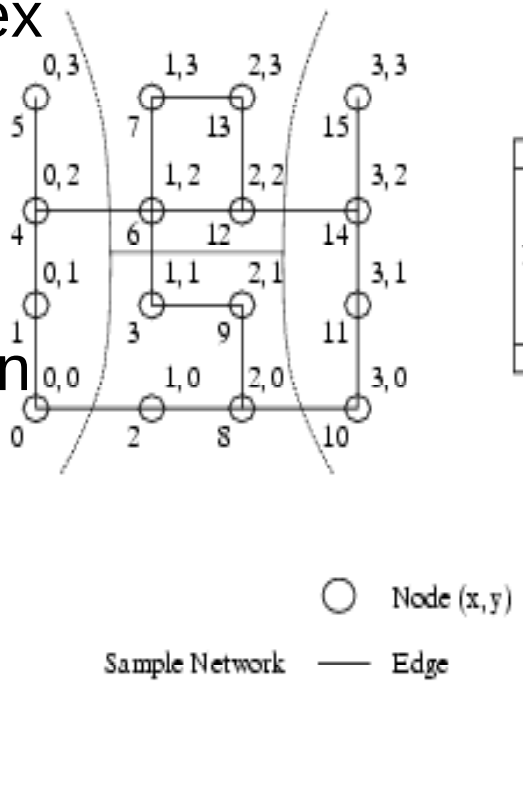
- Consider two disk-paging of Minneapolis (明尼阿波利斯) major roads
  - Non-white edges => node pair in same page
  - White edge are cut-edges
  - Node partitions on right has fewer cut-edges and is preferred => higher CRR

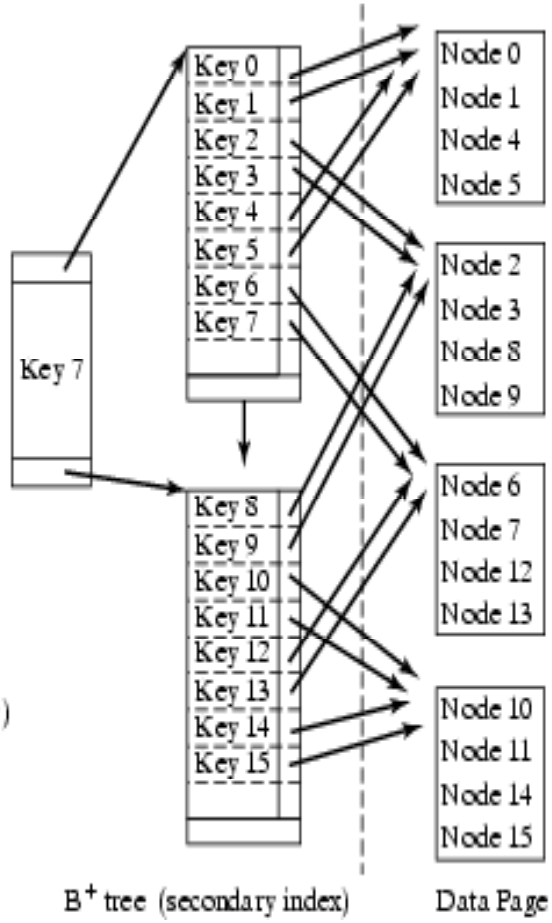




Clustering and Storing a Sample Network

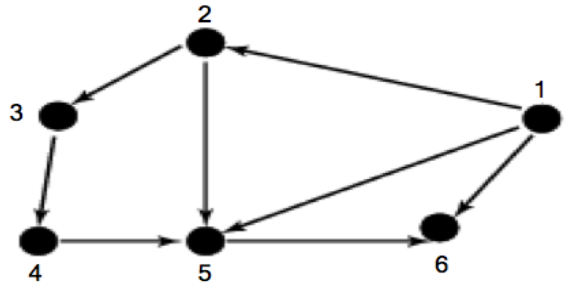
- Storage method idea
  - Divide nodes into sectors
    - to maximize CRR
  - Use a secondary index
    - for find()
    - using R-tree or B-tree
- Example
  - left part : node division
  - right part
    - disk sectors
    - secondary index
    - B-tree/Z-order





Exercise: Graph Based Storage Methods

- If a disk page holds 3 records, which partitioning will has fewest cut-edges?
  - (1, 2, 3), (4, 5, 6)
  - (2, 3, 4), (1, 5, 6)
  - (1, 2, 6), (3, 4, 5)
  - (1, 3, 5), (2, 4, 6)



Node

nid	x	y	Successors	Predecessors
1	—	—	(2,5,6)	()
2	—	—	(3,5)	(1)
3	—	—	(4)	(3)
4	—	—	(5)	(3)
5	—	—	(6)	(2,1)
6	—	—	()	(1,5)

Quiz 8

- Which of the following is not disk-based representations of graphs?
  - a) Normalized tables (e.g., node table and edge table)
  - b) Denormalized table (e.g., node table with successors and predecessors columns)
  - c) Adjacency matrix
  - d) All of the above

9.4.2 Query Processing for Spatial Networks

- Query Processing
  - DBMS decomposes a query into building blocks
  - Keeps a couple of strategy for each building block
  - Selects most suitable one for a given situation

- Building blocks
  - Connectivity(A, B): Is node B reachable from node A?
  - Shortest path(A, B): Identify the least cost path from node A to node B

Algorithms

- Main memory (数据结构基础)
  - Connectivity: Breadth first search, Depth first search
  - Shortest path: Dijkstra's algorithm, A*
    - Estimated cost in A* for spatial data: euclidean distance
- Disk-based
  - Shortest path - Hierarchical routing algorithm
  - Connectivity strategies are in SQL3

Quiz 9

- Which of the following is false?
  - a) Breadth first search visits nodes layer (i.e. generation) by layer
  - b) Depth first search try a path till dead-end, then backtrack to try different paths
  - c) Depth first search always performs better than breadth first search
  - d) None of the above

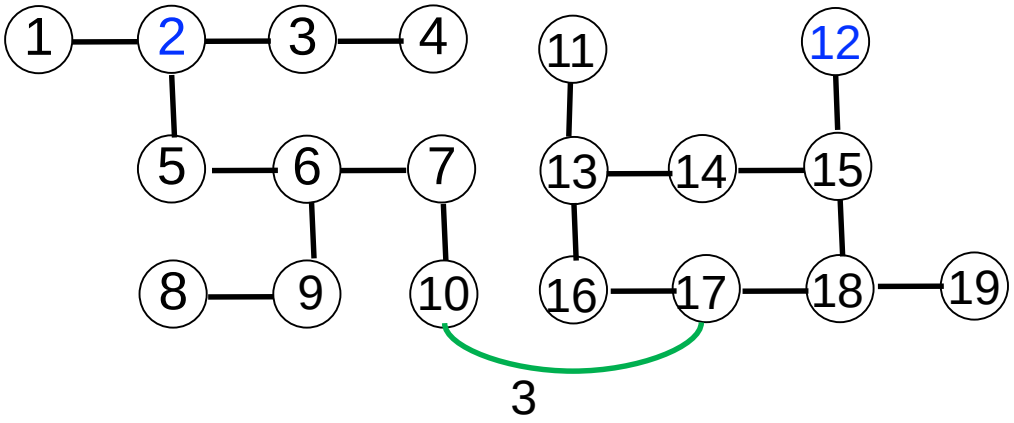
Shortest Path Strategies

- Dijkstra'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 memory
  - Can compute least cost routes

Hierarchical Routing: Simple Example

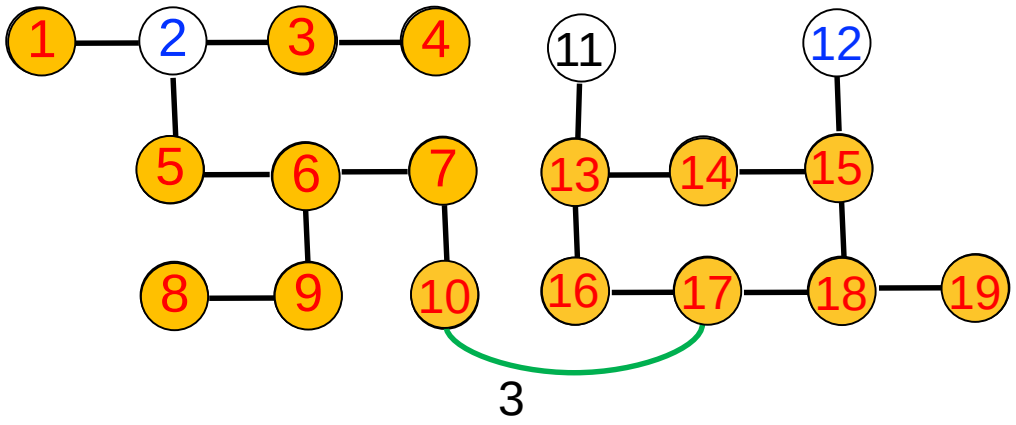
- Goal: Find the shortest path between Nodes 2 and 12
- Candidate algorithms: Dijkstra's algorithm, A*, and hierarchical routing



The edge length is 1 for every edge, except length(edge (11-18)) = 3

Trace Dijkstra's algorithm

- Tie-braking: prefer node with higher index



Q? Will A* (with Euclidean distance) be efficient?

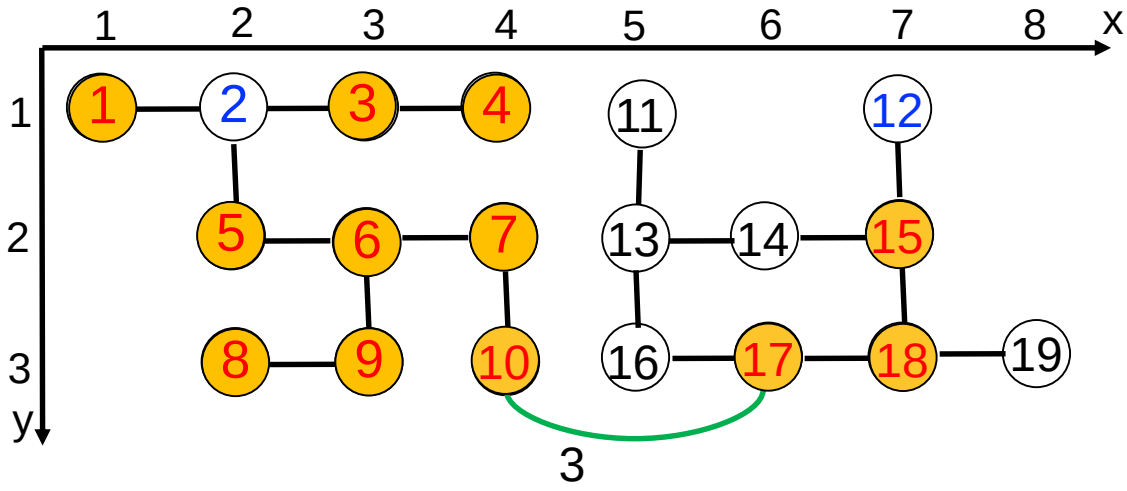
18 nodes expanded

Expanded:	Open List:
2	1, 3, 5
5	1, 3, 6
3	1, 4, 6
1	4, 6
6	4, 7, 9
4	7, 9
9	7, 8
7	8, 10
10	8, 17
8	17
17	16,18
18	15,16,19
16	13,15,19
19	13,15
15	12,13,14
14	12,13
13	11,12
12	

Trace for A* algorithm

Cost(node n) = graph_distance (2, n) + Euclidean_distance (n, 12)

Tie-braking: prefer node with higher index



14 nodes expanded

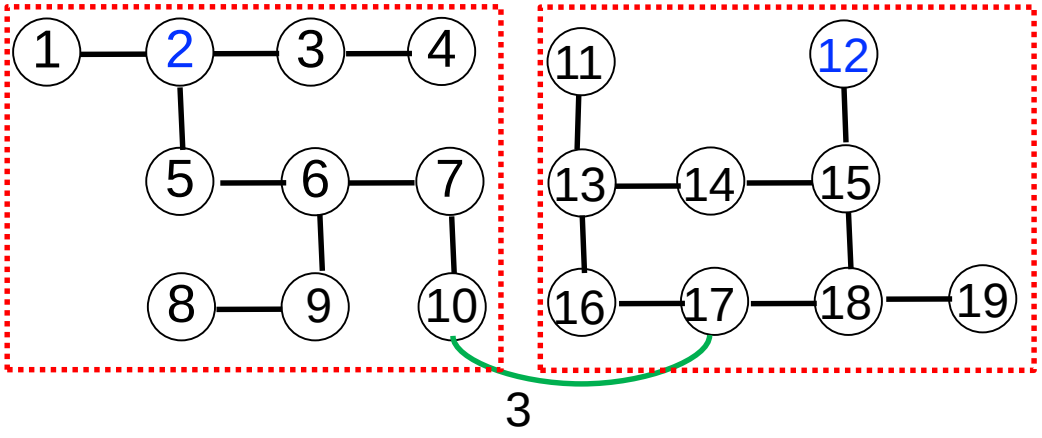
Expand:	Open List:
2	1, 3, 5
3	1, 4, 5
4	1, 5
5	1, 6
6	1, 7, 9
7	1, 9,10
10	1,9,17
1	9,17
9	8,17
17	8, 16,18
8	16,18
18	15,16,19
15	12,14,16,19
12	

Q? How may we reduce computation cost further?

Core Idea of Hierarchical Routing

- Recognize Islands and bridges
- SP(2,12) must include bridge edge (10,17)
- Divide n conquer

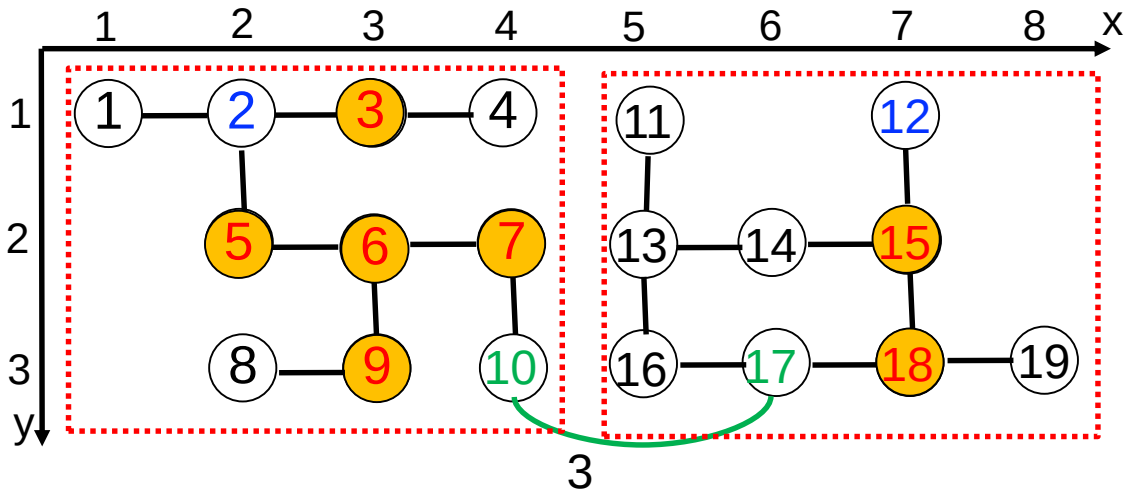
  SP(2,12) = SP(2, 10) + edge(10,17) + SP(17, 12)
- Generalize to the case of multiple bridges



Trace Hierarchical Routing

4 nodes expanded in SP(17 , 12) using A*

Expanded:	Open List:
17	16,18
18	15,16,19
15	12,14,16,19
12	



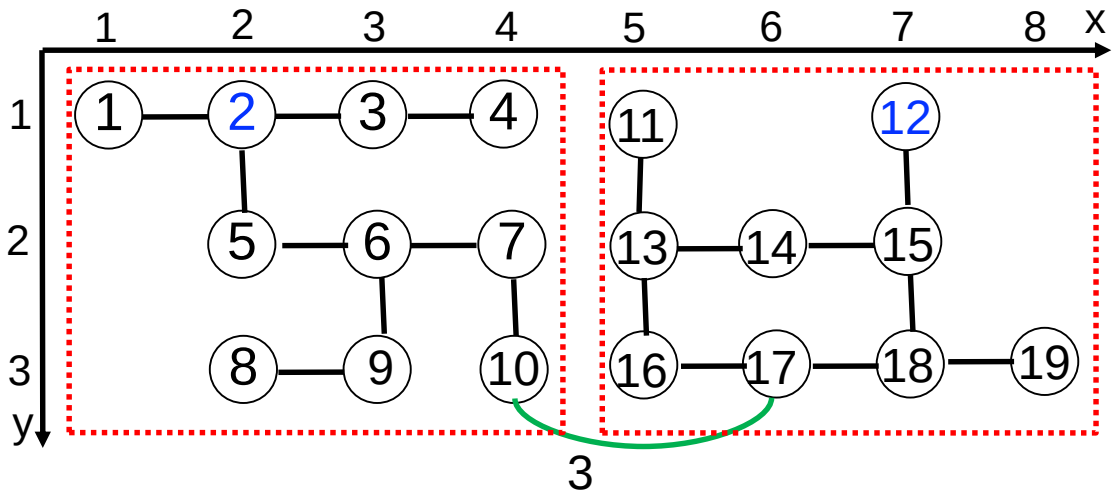
#Nodes expanded: 7 + 4 = 11

7 nodes expanded in SP(2 , 10) using A*

Expanded:	Open List:
2	1,3,5
5	1,3,6
3	1,4,6
6	1,4,7,9
9	1,4,7,8
7	1,4,8,10
10	

Did We Reduce Computational Cost?

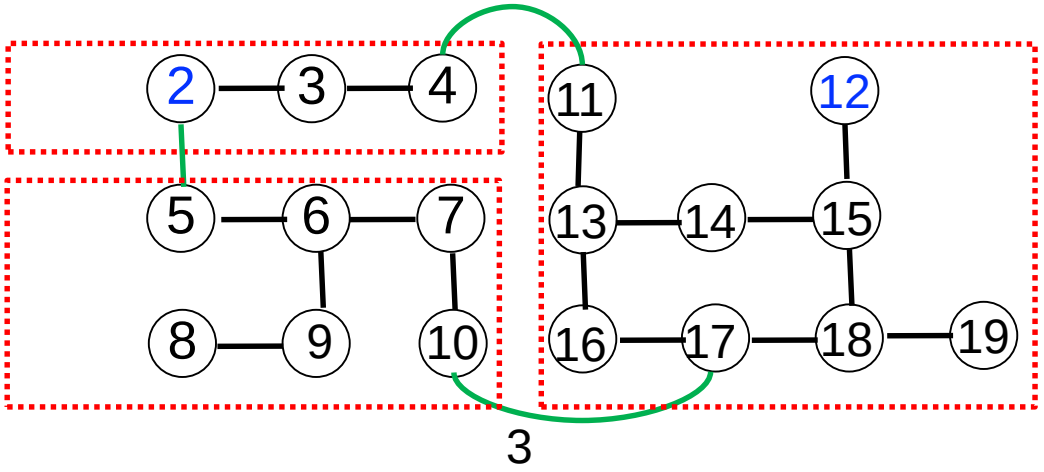
Algorithm (World View)	# of nodes expanded
Dijkstra's (Graphs)	18
A* (Spatial Graphs)	14
Hierarchical Routing (Islands)	7 + 4 = 11



Q? What if Multiple bridges?
Q? How to choose among bridges?

Challenges: Multiple Islands & Bridges

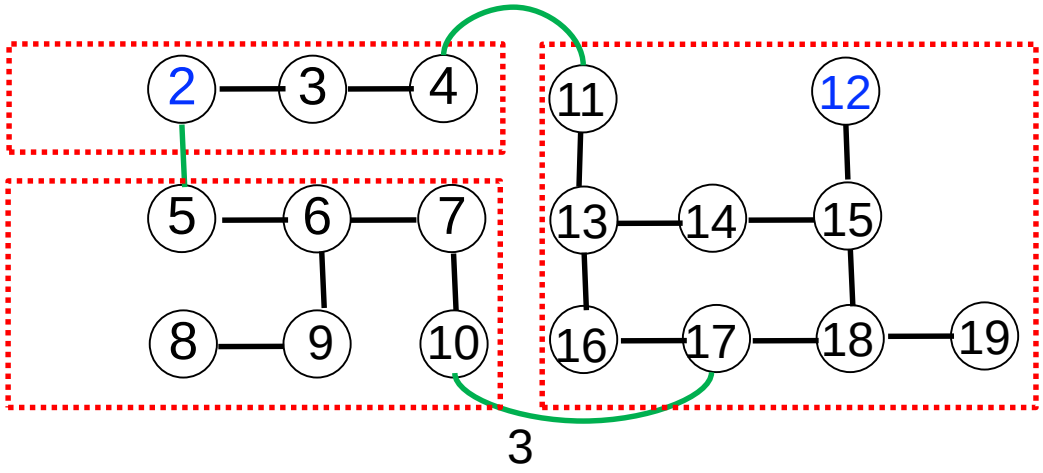
- Invariant: SP(2,12) must use one or more bridges
- Challenge 1: Multiple islands increase computational cost
- Challenge 2: Multiple bridges per island increase computational cost

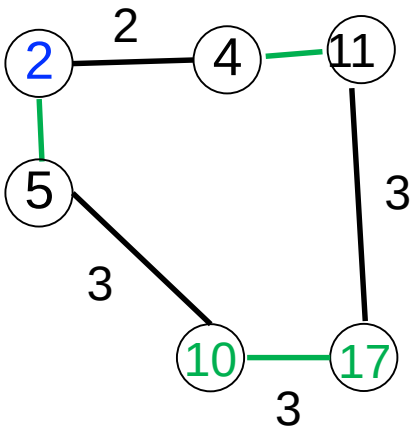


Hierarchical Algorithm with Multiple Islands

Data Structures:

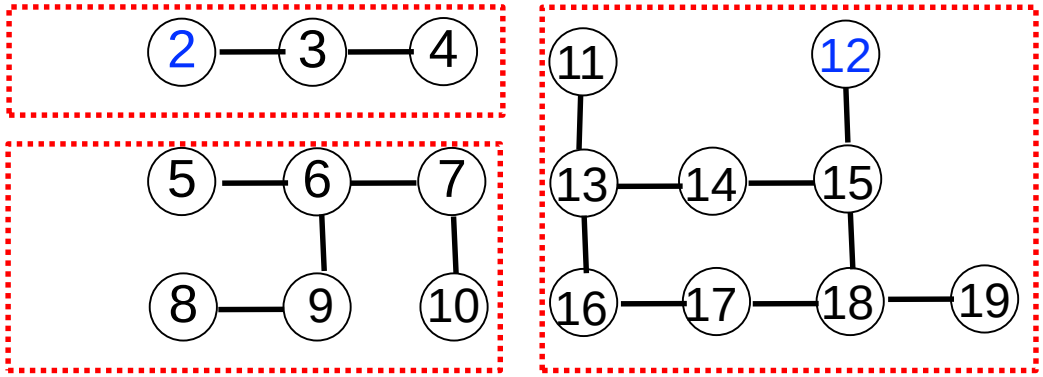
- Port node (a.k.a., boundary node) : a node with edges to multiple islands
- Port Graph (a.k.a., boundary graph) :
- Island graphs (a.k.a., fragment graphs)
- Precompute & store
  - Shortest_path_costs: SPC(node i, node j) for all (or selected) node pairs (j,j)

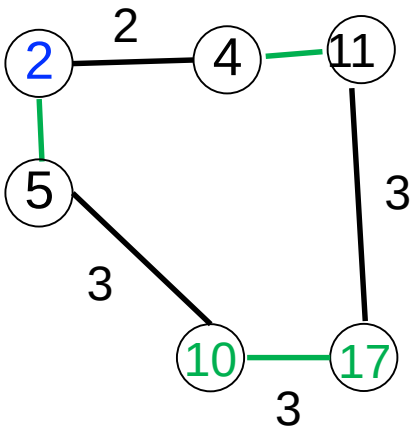




Hierarchical Algorithm with Multiple Islands

- Choose port pair (a.k.a., boundary node pair) for SP(2,12)
  - min SPC(2, local port) + SPC(2's local port, 12's local port) + SPC (local port, 12)
  - Choices: (2,11), (2, 17), (4, 11), (4, 17)
  - Chosen port pair: (4, 11)
- Divide and conquer: SP(2,12) = SP(2, 4) . SP(4, 11) . SP(11, 12)
  - Use Dijktra's or A* for sub-problems
- Refine algorithm to reduce storage cost





Hierarchical Routing: Key Ideas

- 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

Hierarchical Routing: 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 a fragment overlapping with the path
  - Loading one fragment in memory at a time

Quiz 10

- Which of the following is false?
  - a) Hierarchical routing algorithms are Disk-based shortest path algorithms
  - b) Breadth first search and depth first search are both connectivity query algorithms
  - c) Best first algorithm is always faster than Dijkstra's algorithm
  - d) None of the above

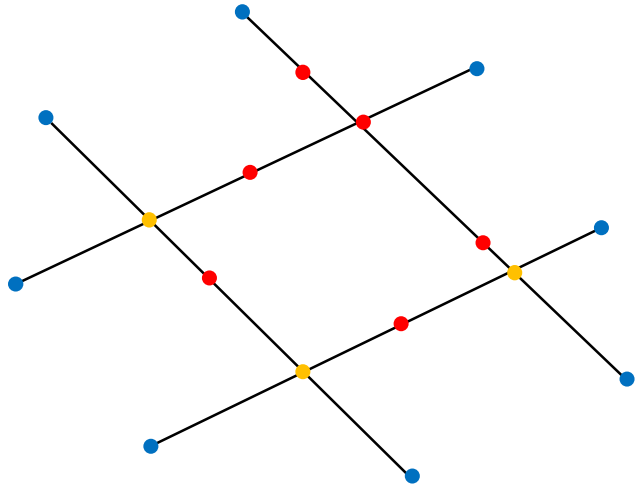
第九章 空间网络模型与查询

- 9.1 Motivation and use cases
- 9.2 Conceptual model
- 9.3 Logical model
  - 9.3.1 Transitive Closure
  - 9.3.2 CONNECT statement
  - 9.3.3 RECURSIVE statement
  - 9.3.4 RECURSIVE in PostgreSQL
- 9.4 Physical model
  - 9.4.1 Storage and data structures
  - 9.4.2 Algorithms for connectivity query and shortest path
- 9.5 pgRouting

参考教材：
Spatial Databases: A Tour Chapter 6
空间数据库管理系统概论，3.3-3.4

几何对象模型 → 空间网络模型

- 节点选择
  - 所有顶点 ✗
    - 网络节点过多
  - 线段端点
    - 一条道路仅两个端点
  - 线段交点
    - 增加节点，增强网络连通性
  - 距离相近的顶点合并
    - 减少节点，增强网络连通性
- 边选择
  - 节点之间的道路
  - 有向图、无向图



pgRouting

- pgRouting - geospatial routing functionality
- Advantages of the database routing approach
  - Data and attributes can be modified by many clients, like QGIS through JDBC, ODBC, or directly using Pl/pgSQL
  - Data changes can be reflected instantaneously through the routing engine
  - The “cost” parameter can be dynamically calculated through SQL and its value can come from multiple fields or tables
- How to create a database and load pgRouting
  - createdb netdatabase
  - psql netdatabase -c “create extension postgis”
  - psql netdatabase -c “create extension pgrouting”

http://docs.pgrouting.org/

pgRouting

- How to build a topology
  - Ends of an edge will be connected to a unique node
  - Select pgr_createTopology(‘myroads’, 0.000001)
  - 线段端点作为网络节点，合并距离相近的节点，距离相近判断阈值为参数0.000001，构建网络
- How to check your graph for errors
  - Select prg_analyzeGraph(‘myroads’, 0.000001);
  - Select prg_analyzeOneway(‘myroads’, s_in_rules, s_out_rules, t_in_rules, t_out_rules, direction);
  - 分析当前网络是否存在距离相近的节点，是否存在线段交点

pgRouting

- How to build a topology
  - Select pgr_createTopology(‘myroads’, 0.000001)
- How to check your graph for errors
  - Select prg_analyzeGraph(‘myroads’, 0.000001);
- How to renode if there is any error
  - pgr_nodeNetwork
  - 将相交的线段按交点分成多条线段
- How to compute a route
  - Select pgr_<algorithm> (<SQL for edges>, start, end, <additional options>
    - pgr_dijkstra(text edges_sql, bigint start_vid, bigint end_vid, boolean directed:=true)
  - pgr_costResult[] and pgr_geomResult[]

pgRouting Data Types

- pgr_costResult[]
  - seq integer -- sequential ID indicating the path order
  - id1 integer -- typically the node id
  - id2 integer -- typically the edge id
  - cost float8   -- cost attribute
- pgr_geomResult[]
  - seq integer -- sequential ID indicating the path order
  - id1 integer -- generic name, to be specified by the func
  - id2 integer -- generic name, to be specified by the func
  - geom geometry

pgRouting Topology Functions

- Topology of a network
  - An edge table with source and target attributes
  - A vertices table associated with the edge table
- pgr_createTopology
  - Build a network topology based on the geometry information
  - Parameters
    - edge_table text - Newtork table name
    - tolerance float8 - Snapping tolerance of disconnected edge
    - the_geom text - Geometry colum name of the network table
    - id text - Primary key column name of the network table
    - source text - Source column name of the network table
    - target text - Target column name of the network table
    - rows_where text - Condition to SELECT a subset or rows

pgRouting Topology Functions

- pgr_createTopology
  - Builds a network topology based on the geometry information
  - The edge_table will be affected
    - The source column values will change
    - The target column values will change
    - An index will be created, if it doesn't exists, to speed up the process to the following columns
      - id, the_geom, source, target
  - OK after the network topology has been built
    - Creates a vertices table: <edge_table>_vertices_pgr
    - Fills id and the_geom columns of the vertices table
    - Fills the source and target columns of the edge table referencing the id of the vertices table

pgRouting Topology Functions

- pgr_createTopology
  - Builds a network topology based on the geometry information
  - The edge_table will be affected
  - OK after the network topology has been built
  - FAIL when the network topology was not built due to an error
    - A required column of the Network table is not found or is not of the appropriate type
    - The condition is not well formed
    - The names of source, target or id are the same
    - The SRID of the geometry could not be determined

pgRouting Topology Functions

- pgr_createTopology
  - The Vertices Table
    - id bigint - Identifier of the vertex.
    - cnt integer - Number of vertices in the edge_table that reference this vertex
    - chk integer - Indicator that the vertex might have a problem
    - ein integer - Number of vertices in the edge_table that reference this vertex AS incoming
    - eout integer - Number of vertices in the edge_table that reference this vertex AS outgoing
    - the_geom geometry - Point geometry of the vertex
- prg_createVerticesTable
  - Reconstructs the vertices table based on the source and target information

pgRouting Topology Functions

id	the_geom	source	target	…
81	(0 20, 10 20.01)			
45	(10 19.99, 10 2)			
…	…			

几何表
edge_table

pgr_createTopology
tolerance =0.05

边表 edge_table

id	the_geom	source	target	…
81	(0 20, 10 20.01)	3	5	
45	(10 19.99, 10 2)	5	7	
…	…	…	…	

顶点表 edge_table_vertices_pgr

id	cnt	chk	ein	eout	the_geom
3					(0 20)
5					(10 20)
…					..

pgRouting Topology Functions

- pgr_analyzeGraph
  - Analyze the network topology
  - The edge table to be analyzed must contain
    - a source column and a target column filled with id's of the vertices of the segments
    - the corresponding vertices table <edge_table>_vertices_pgr that stores the vertices information

```
varchar pgr_analyzeGraph(text edge_table, double precision tolerance,
                  text the_geom:='the_geom', text id:='id',
                  text source:='source',text target:='target',text rows_where:='true')
```

- pgr_analyzeOneway
  - Analyzes oneway streets and identifies flipped segments

pgRouting Topology Functions

- pgr_analyzeGraph
  - Analyzes the network topology
    - edge_table: text Network table name. (may contain the schema name as well)
    - tolerance: float8 Snapping tolerance of disconnected edges. (in projection unit)
    - the_geom: text Geometry column name of the network table. Default value is the_geom
    - id: text Primary key column name of the network table. Default value is id
    - source: text Source column name of the network table. Default value is source
    - target: text Target column name of the network table. Default value is target
    - rows_where: text Condition to select a subset or rows. Default value is true to indicate all rows

pgRouting Topology Functions

- pgr_analyzeGraph
  - Analyzes the network topology
  - OK after the analysis has finished
    - Uses the vertices table: <edge_table>_vertices_pgr
    - Fills compeletly the cnt and chk columns of the vertices table
    - Returns the analysis of the section of the network defined by rows_where
  - FAIL when the analysis was not completed due to an error
    - The vertices table is not found
    - A required column of the Network table is not found or is not of the appropriate type
    - The condition is not well formed
    - The names of source, target or id are the same
    - The SRID of the geometry could not be determined

pgRouting Topology Functions

边表 edge_table

id	the_geom	source	target	…
81	(0 20, 10 20)	3	5	
45	(10 20, 10 2)	5	7	
…	…	…	…	

顶点表 edge_table_vertices_pgr

id	cnt	chk	ein	eout	the_geom
3					(0 20)
5					(10 20)
…					..

pgr_analyzeGraph
tolerance =0.05

顶点表 edge_table_vertices_pgr

id	cnt	chk	ein	eout	the_geom
3	2	0			(0 20)
5	4	1			(10 20)
…	…	…			..

pgRouting Topology Functions

- pgr_nodeNetwork
  - Node an network edge table
  - This function reads the edge_table table, that has a primary key column id and geometry column named the_geom and intersect all the segments in it against all the other segments and then creates a table edge_table_noded
  - It uses the tolerance for deciding that multiple nodes within the tolerance are considered the same node

pgRouting Topology Functions

- pgr_nodeNetwork
  - Node an network edge table
  - edge_table: text Network table name. (may contain the schema name as well)
  - tolerance: float8 tolerance for coincident points (in projection unit)
  - id: text Primary key column name of the network table. Default value is id
  - the_geom: text Geometry column name of the network table. Default value is the_geom
  - table_ending: text Suffix for the new table's. Default value is noded

pgRouting Topology Functions

- pgr_nodeNetwork
  - Nodes an network edge table
  - The output table will have for edge_table_noded
  - id: bigint Unique identifier for the table
  - old_id: bigint Identifier of the edge in original table
  - sub_id: integer Segment number of the original edge
  - source: integer Empty source column to be used with pgr_createTopology function
  - target: integer Empty target column to be used with pgr_createTopology function
  - the_geom: geometry Geometry column of the noded network

pgRouting Topology Functions

边表 edge_table

id	the_geom	source	target	…
81	(0 20, 10 20)			
45	(10 20, 10 2)			
…	…			

pgr_nodeNetwork
tolerance =0.05

边表 edge_table_noded

id	oid	subid	source	target	the_geom
34	81	1			(0 20, 5 20)
35	81	2			(5 20, 10 20)
36	45	1			(10 20, 10 2)
…	…	…			…
…	…	…			…

pgRouting Routing Functions

- All Pairs Shortest Path, Johnson's Algorithm
- All Pairs Shortest Path, Floyd-Warshall Algorithm
- Shortest Path A*
- Bi-directional Dijkstra Shortest Path
- Bi-directional A* Shortest Path
- Shortest Path Dijkstra
- Driving Distance
- K-Shortest Path, Multiple Alternative Paths
- K-Dijkstra, One to Many Shortest Path
- Traveling Sales Person
- Turn Restriction Shortest Path (TRSP)

http://docs.pgrouting.org/

Dijkstra最短路径算法

- pgr_dijkstra(text edges_sql, bigint start_vid, bigint end_vid, boolean directed:=true)
  - edges_sql: a SQL query
  - select id, source, target, cost [, reverse_cost] from edge_table
    - id: ANY-INTEGER identifier of the edge
    - source: ANY-INTEGER identifier of the first end point vertex of the edge
    - target: ANY-INTEGER identifier of the second end pont vertex of the edge
    - cost: ANY-NUMERICAL weight of the edge (source, target), if negative: edge (source, target) does not exist, therefore it's not part of the graph
    - reverse_cost: ANY-NUMERICAL (optional) weight of the edge (target, source), if negative: edge (target, source) does not exist, therefore it's not part of the graph

Dijkstra最短路径算法

- pgr_dijkstra(text edges_sql, bigint start_vid, bigint end_vid, boolean directed:=true)
  - edges_sql: a SQL query
    - select id, source, target, cost [, reverse_cost] from edge_table
  - start_vid: BIGINT identifier of the starting vertex of the path
  - end_vid: BIGINT identifier of the ending vertex of the path
  - directed: boolean (optional). When false the graph is considered as Undirected. Default is true which considers the graph as Directed

Dijkstra最短路径算法

- pgr_costResult[] pgr_dijkstra(text sql, interger source, integer target, boolean directed:=true)

```
SELECT * FROM pgr_dijkstra(
        'SELECT id, source, target, cost, reverse_cost FROM edge_table',
        2, 3
);
 seq | path_seq | node | edge | cost | agg_cost
-----+----------+------+------+------+----------
   1 |        1 |    2 |    4 |    1 |        0
   2 |        2 |    5 |    8 |    1 |        1
   3 |        3 |    6 |    9 |    1 |        2
   4 |        4 |    9 |   16 |    1 |        3
   5 |        5 |    4 |    3 |    1 |        4
   6 |        6 |    3 |   -1 |    0 |        5
(6 rows)


SELECT * FROM pgr_dijkstra(
        'SELECT id, source, target, cost, reverse_cost FROM edge_table',
        2, 5
);
 seq | path_seq | node | edge | cost | agg_cost
-----+----------+------+------+------+----------
   1 |        1 |    2 |    4 |    1 |        0
   2 |        2 |    5 |   -1 |    0 |        1
(2 rows)
```

Dijkstra最短路径算法

- SELECT seq, id1 AS node, id2 AS edge, cost
  FROM pgr_dijkstra(
    'SELECT id, source, target, len as cost FROM road_network',
    5, 3, false
  );
  - 通过SQL语句构建查询表，提供网络关系(id, source, target, cost)
  - 查询约束，通过where语句查询不同网络关系实现
    - 某条道路在修路: where id != 101
    - 只走cost小于10的道路：where cost < 10
    - ….

Example

- create table Road (
  id serial primary key,
  name text,
  geom geometry(LineString, 4326));

- insert into Road(name, geom) values
- ('A', ST_GeomFromText('LineString(0 20, 10 20)', 4326));
- ('B', ST_GeomFromText('LineString(10 20, 10 2)', 4326));
- ('C', ST_GeomFromText('LineString(10 2, 20 2)', 4326));
- ('D', ST_GeomFromText('LineString(0 5, 20 5)', 4326));
- ('E', ST_GeomFromText('LineString(20 4.9999, 20 2.0001)', 4326));

Example

- create table Road_network (
  id serial primary key,
  name text,
  source int,
  target int,
  geom geometry(LineString, 4326),
  len float);

- insert into Road_network(name, geom) select name, geom from Road;

- select pgr_createTopology(‘road_network', 0.00001, 'geom', 'id', 'source', 'target', 'true');

- select pgr_analyzeGraph(‘road_network', 0.00001, 'geom', 'id', 'source', 'target', 'true');

Example

- select pgr_nodeNetwork(‘road_network', 0.00001, the_geom:='geom', id:='id', table_ending:='1');
- select pgr_createTopology(‘road_network_1', 0.001, 'geom', 'id', 'source', 'target', 'true');
- select pgr_analyzeGraph(‘road_network_1', 0.001, 'geom', 'id', 'source', 'target', 'true');

Example

- create table temp (
  id serial primary key,
  name text,
  source int,
  target int,
  geom geometry(LineString, 4326));

- insert into temp(name, source, target, geom)
  select O.name, N.source, N.target, N.geom from Road_network O, Road_network_1 N where O.id = N.old_id order by O.name;

- delete from Road_network;

- insert into Road_network(name, source, target, geom)
  select name, source, target, geom from temp;

Example

- update Road_network
  set len = ST_LENGTH(geom);

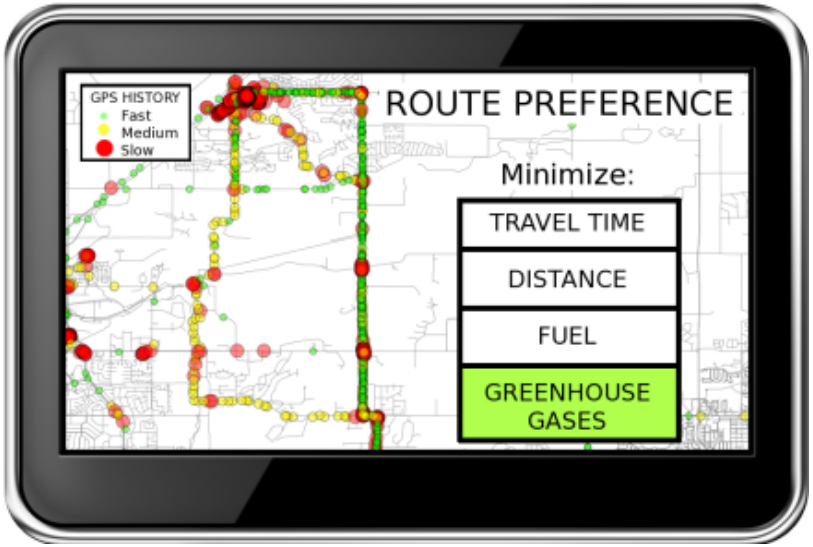
- SELECT seq, id1 AS node, id2 AS edge, cost
  FROM pgr_dijkstra(
  'SELECT id, source, target, len as cost FROM Road_network',
  5, 3, false
  );

Example

- 扩展
  - 新增加路F，从(0,5)到(10,20)
  - 查询(12,6)到(10,20)的最短距离
  - 查询(12,6)到(18,4)的最短距离
  - 限制单行，只能从(10,5)到(0,5)

Trends: Next Generation Navigation Services

- Eco-Routing
  - Energy cost may be negative, e.g., downhill
  - Dijkstra's assumptions
    - Sub-path optimality
    - Fixed, non-negative and additive edge costs
  - Relaxing assumptions
    - Fixed and additive edge cost (maybe negative)
    - No "negative cycle"
- Spatio-temporal Graphs
  - Time variant networks
  - Flow networks



Summary

- Spatial Networks are a fast growing applications of SDBs
- Spatial Networks are modeled as graphs
- Graph queries, like shortest path, are transitive closure
  - Not supported in relational algebra
  - SQL features for transitive closure: CONNECT BY, WITH RECURSIVE, pgRouting
- Graph Query Processing
  - Building blocks - connectivity, shortest paths
  - Strategies - Best first, Dijkstra’s and Hierarchical routing
- Storage and access methods
  - Minimize CRR

空间数据模型总结

- 几何对象模型
- 几何拓扑模型
- 空间网络模型
- 栅格数据模型 (遥感数字图像处理)
- 注记文字模型