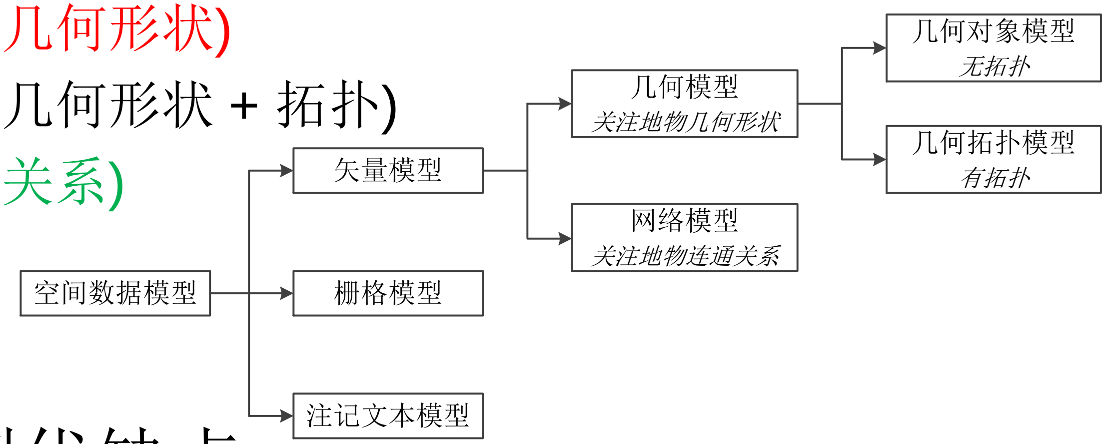
第十一章 空间网络模型与查询

陶煜波
计算机科学与技术学院

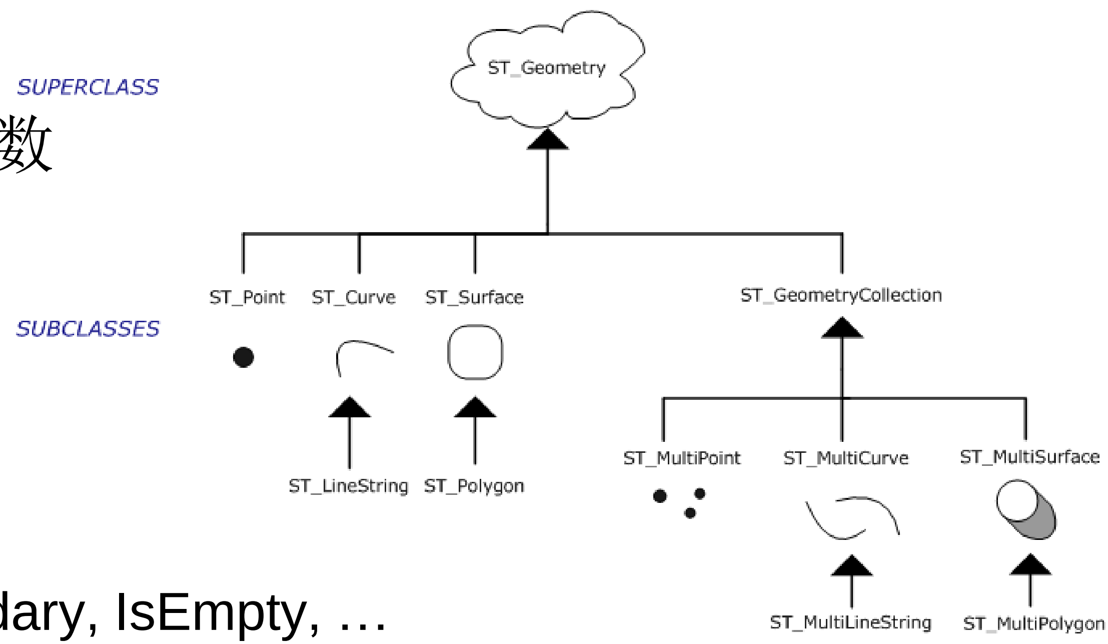
几何对象模型

- 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
- 矢量模型和栅格模型优缺点
- 概念模型 → 逻辑模型 → 物理模型



几何对象模型

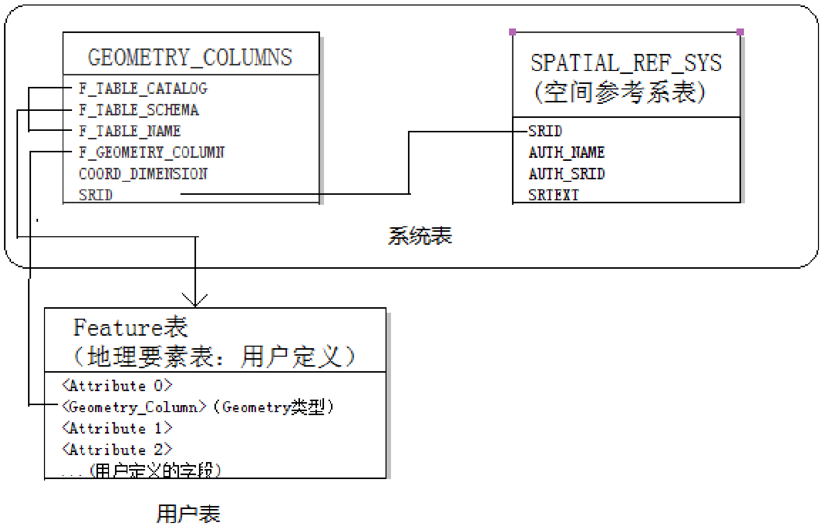
- 对象关系数据库：数据+方法（C++中的类）
- 几何对象层次关系
  - 坐标维数和几何维数
  - 边界、内部、外部
  - 九交矩阵



- 几何对象方法
  - 常规方法 (12种)
    - Dimension, Boundary, IsEmpty, …
  - 常规GIS分析方法 (7种)
    - Distance, Buffer, ConvexHull, Intersection, Union, Difference, SymDifference
  - 空间查询方法 (11种)
    - 包含于(within): 若$a \cap b = a$，且$\mathrm{I}(a) \cap E(b) = \emptyset$，则a包含于b内

几何对象模型

- 逻辑模型
  - 基于预定义数据类型的实现
    - numeric和BLOB
  - 基于扩展几何类型的实现
    - Geometry类
- 表模式
  - 系统表
    - GEOMETRY_COLUMNS和SPATIAL_REF_SYS
  - 用户表
    - Feature和Geometry
- 物理模型
  - WKB和WKT



基于扩展Geometry类型的要素表模式

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
  - Geometry (Point/Line/Polygon/Multixxx, 空间参考系)
  - ST_XXX
  - 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))
- 应如何修改上述SQL语句?
- 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) < 1000

    and T.time ……

    and U.time ……
  - 这样的实现方式合理吗?

第十一章 空间网络模型与查询

- 11.1 Motivation, and use cases
- 11.2 Example spatial networks
- 11.3 Conceptual model
- 11.4 Need for SQL extensions
- 11.5 CONNECT statement
- 11.6 RECURSIVE statement
- 11.7 RECURSIVE in PostgreSQL
- 11.8 Storage and data structures
- 11.9 Algorithms for connectivity query
- 11.10 Algorithms for shortest path

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

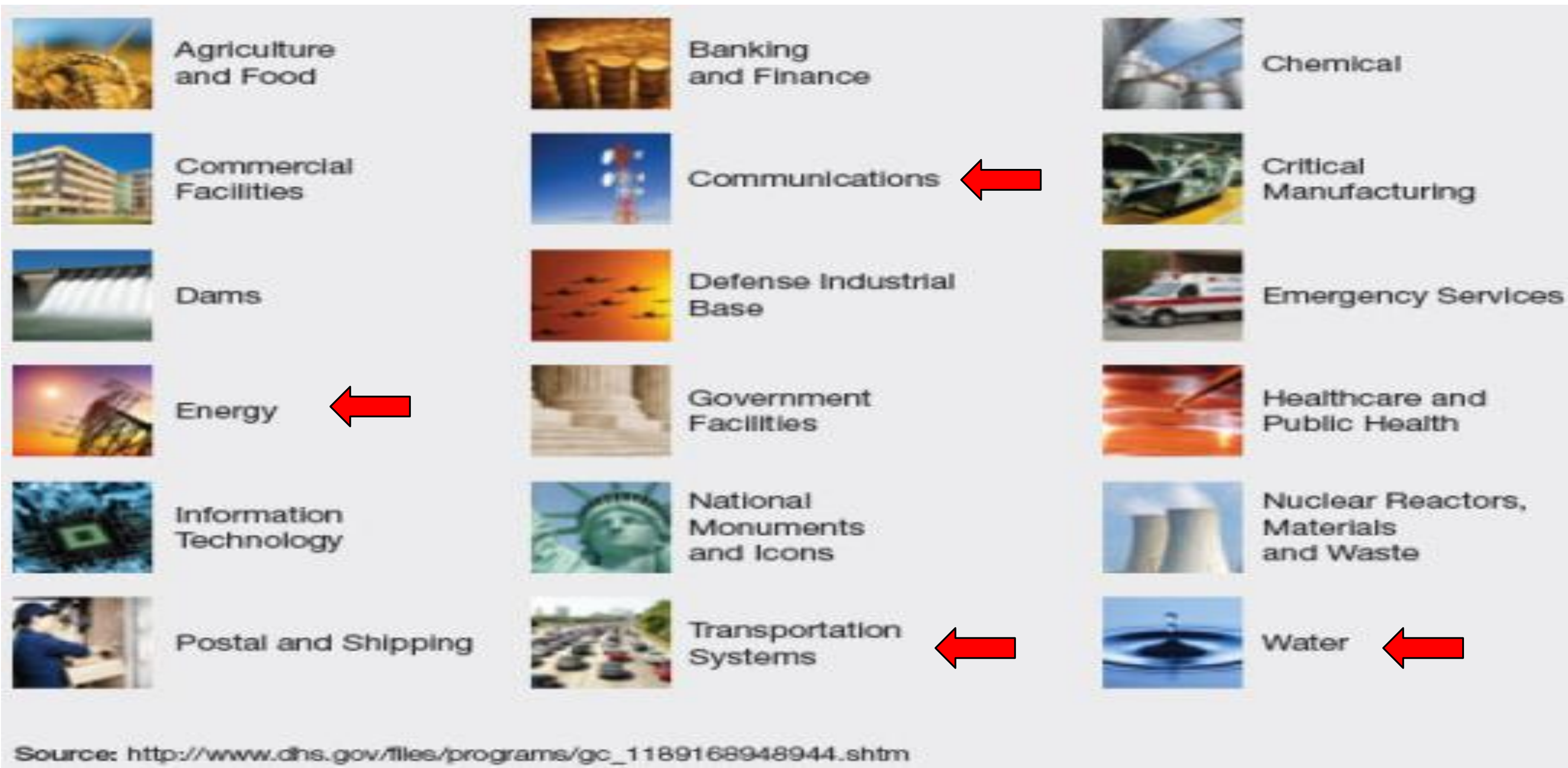
- 休闲娱乐型
  - 签到模式
  - 大富翁游戏模式
- 生活服务型
  - 周边生活服务的搜索
  - 与旅游的结合
  - 会员卡与票务模式
- 社交型
  - 地点交友，即时通讯
  - 以地理位置为基础的小型社区
- 商业型

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

  - LBS+团购；优惠信息推送服务；店内模式

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

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

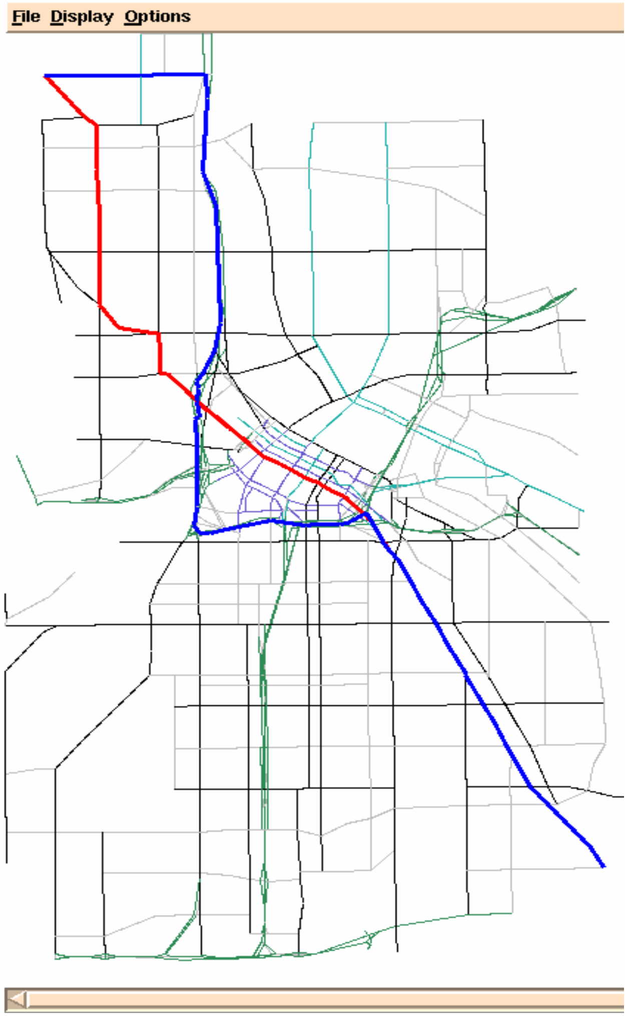
- Which of the following is not included in OGIS simple feature types?
  - a) Graph
  - b) Point
  - c) Line String
  - d) Polygon

第十一章 空间网络模型与查询

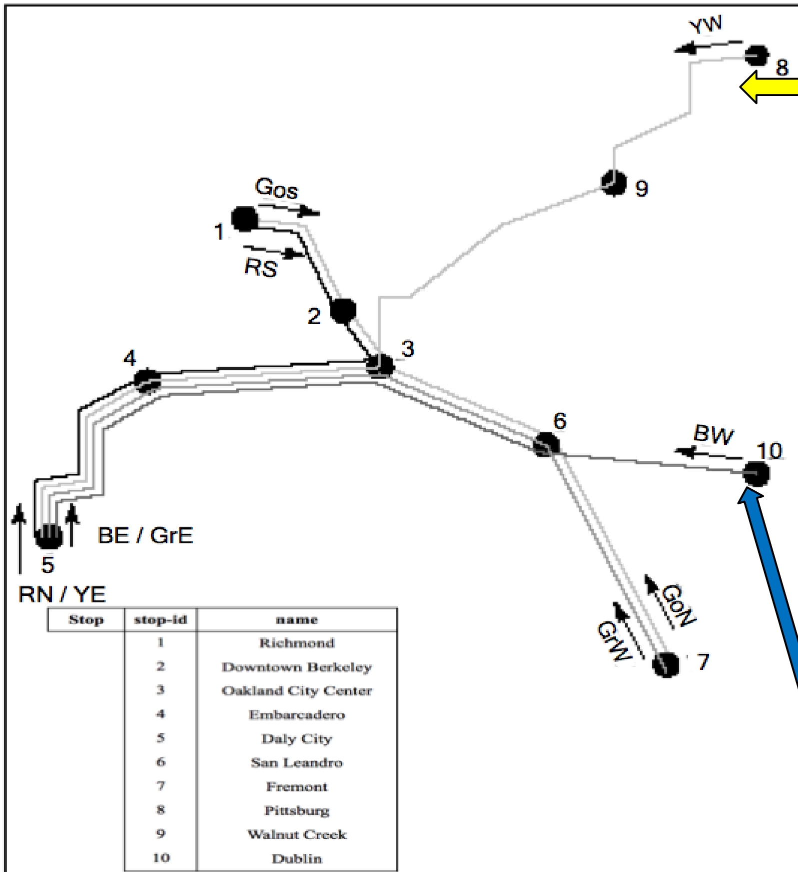
- 11.1 Motivation, and use cases
- 11.2 Example spatial networks
- 11.3 Conceptual model
- 11.4 Need for SQL extensions
- 11.5 CONNECT statement
- 11.6 RECURSIVE statement
- 11.7 RECURSIVE in PostgreSQL
- 11.8 Storage and data structures
- 11.9 Algorithms for connectivity query
- 11.10 Algorithms for shortest path

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

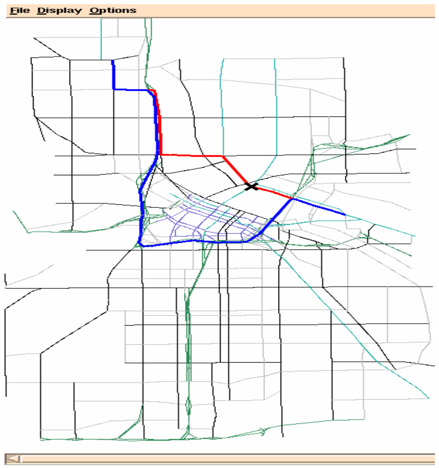
River Network & Queries

- 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

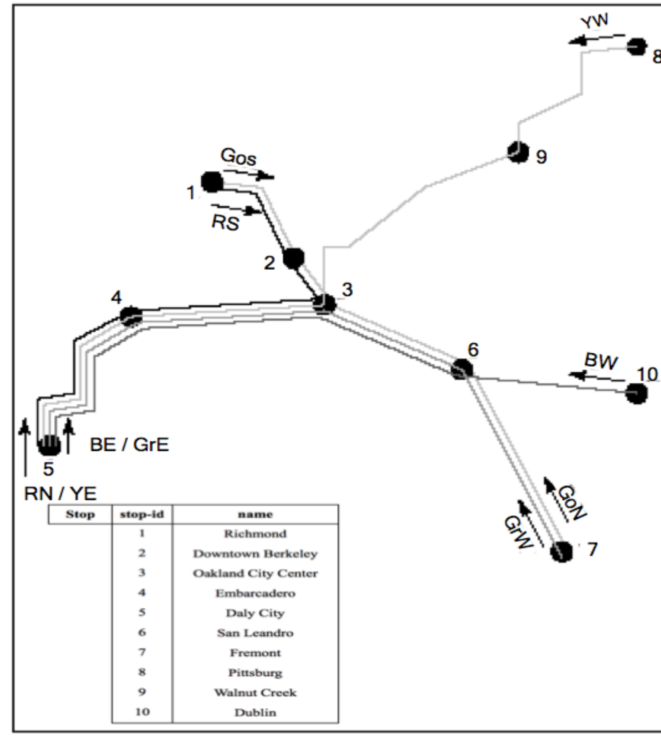
(b) River Network

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

第十一章 空间网络模型与查询

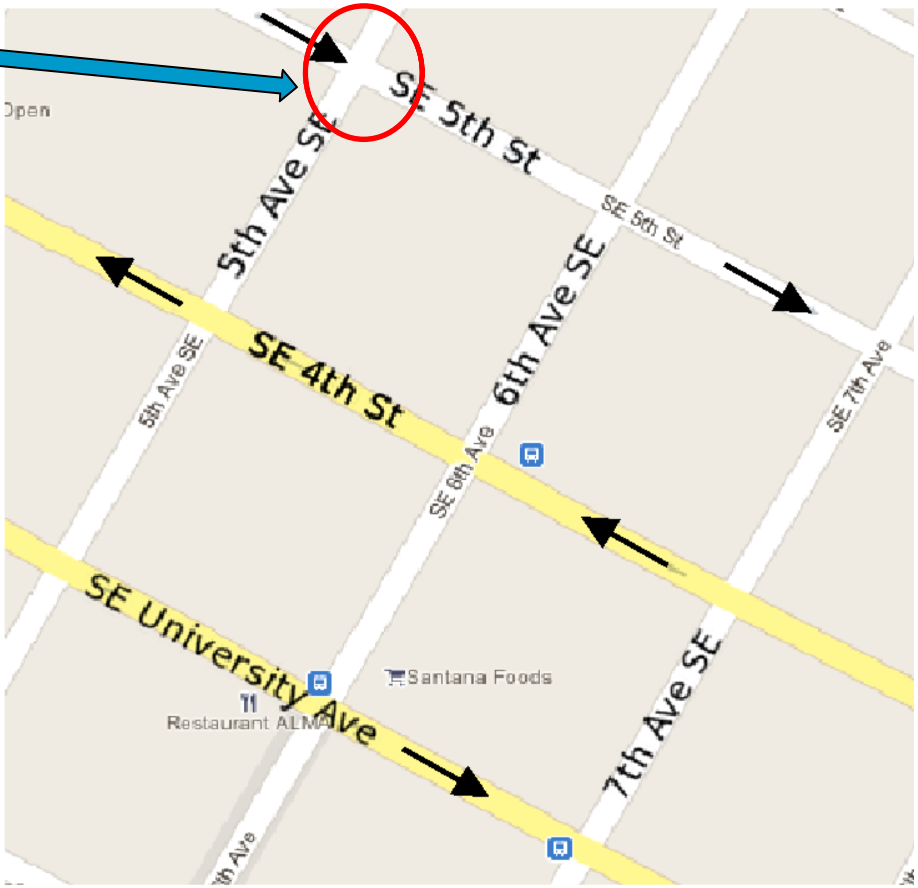
- 11.1 Motivation, and use cases
- 11.2 Example spatial networks
- 11.3 Conceptual model
- 11.4 Need for SQL extensions
- 11.5 CONNECT statement
- 11.6 RECURSIVE statement
- 11.7 RECURSIVE in PostgreSQL
- 11.8 Storage and data structures
- 11.9 Algorithms for connectivity query
- 11.10 Algorithms for shortest path

Data Models of Spatial Networks

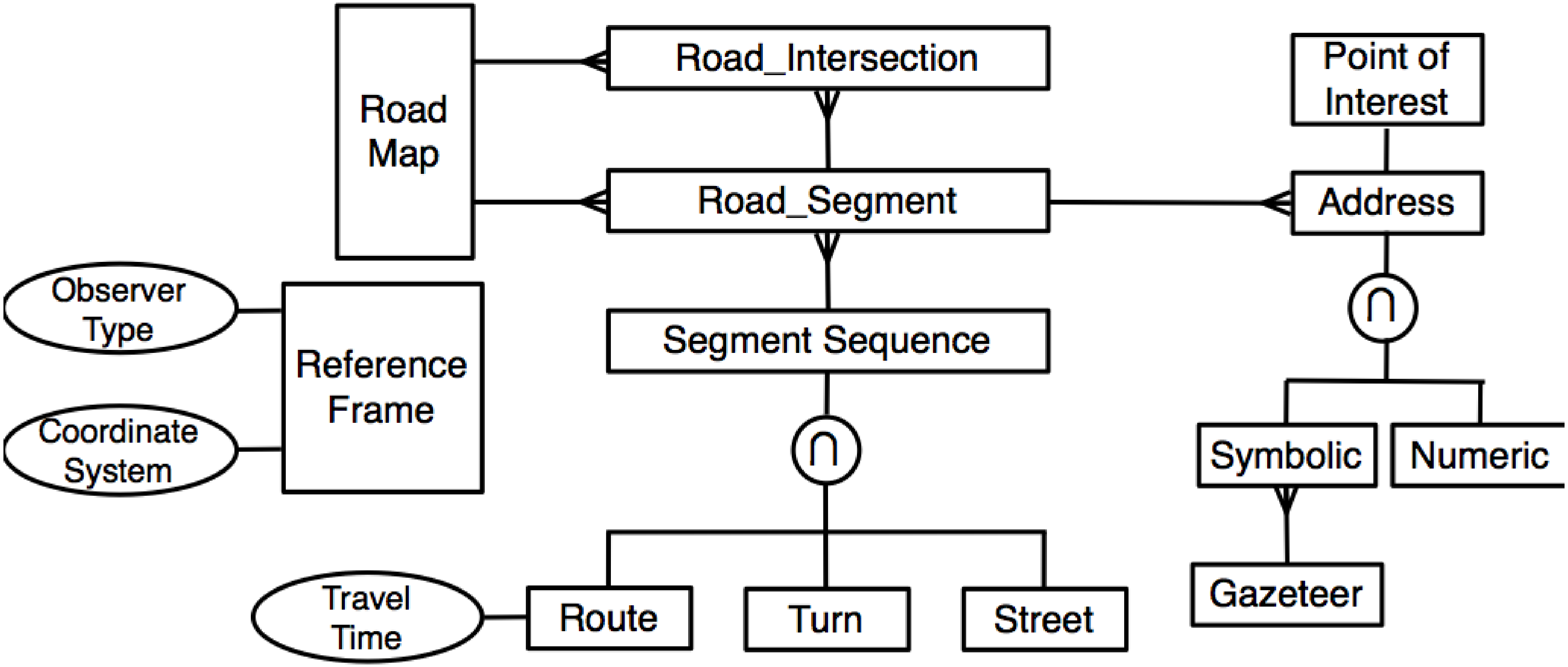
- Conceptual Model
  - Information Model: Entity Relationship Diagrams
  - Mathematical Model: Graphs
- Logical Data Model
  - Abstract Data types
  - Custom Statements in SQL
- Physical Data Model
  - Storage-Structures, File-Structures
  - Algorithms for common operations

Modeling Roadmaps

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

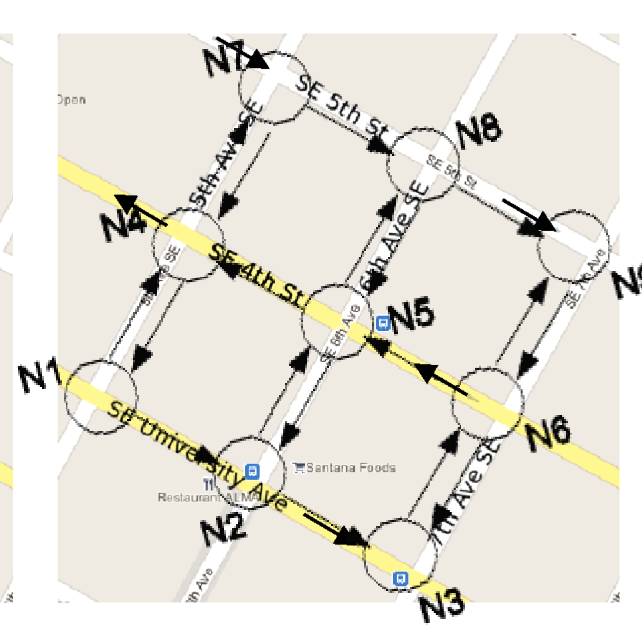


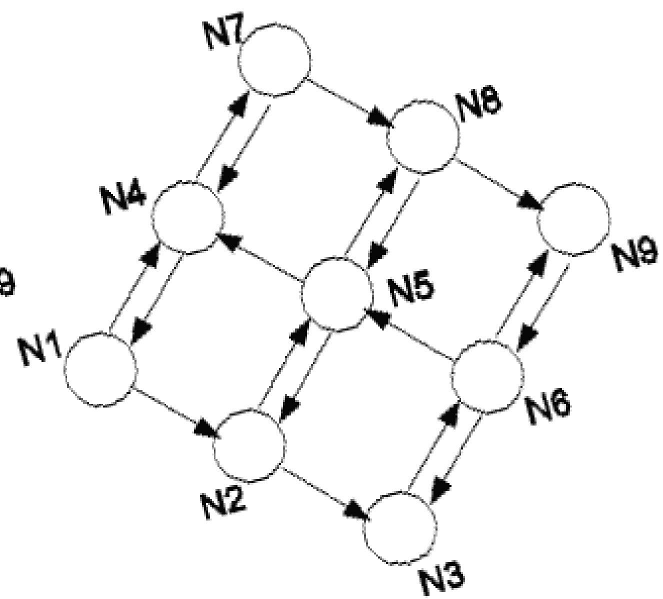
An Entity Relationship Diagram



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

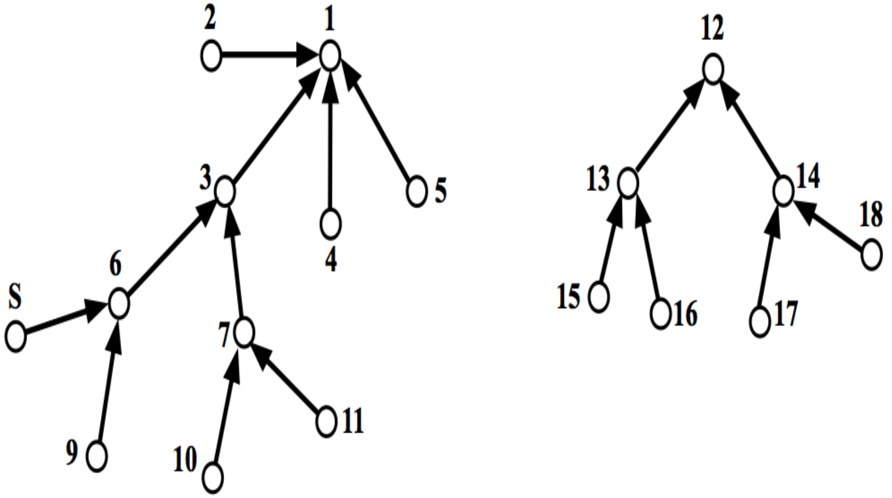




A Graph Model of River Network

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

(b) River Network



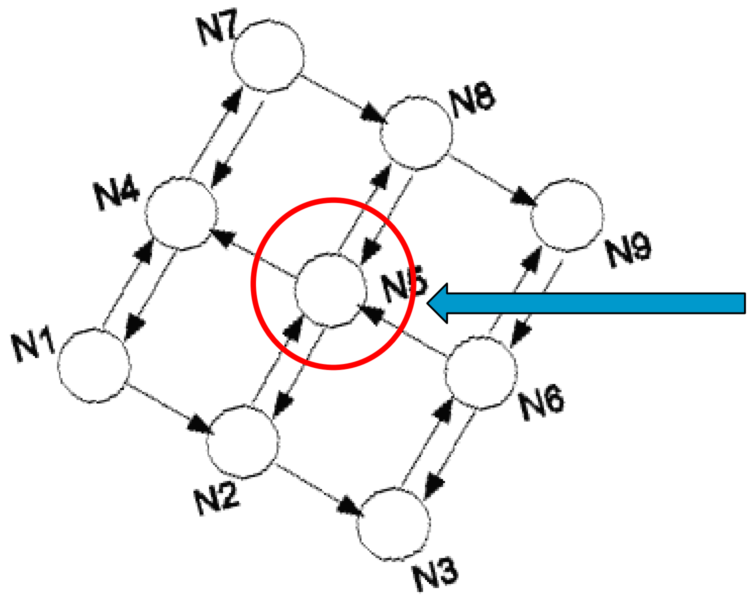
Pros and Cons of Graph Models

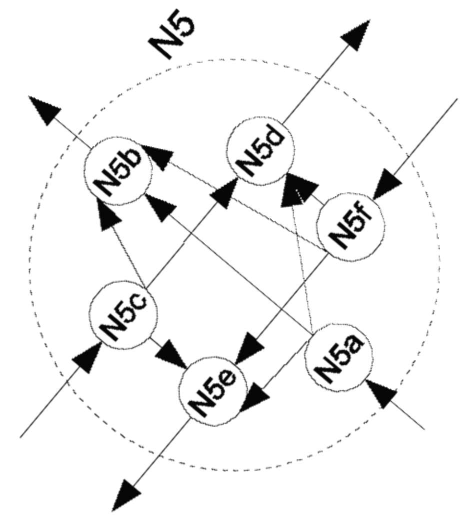
- Strength
  - Well developed mathematics for reasoning
  - Rich set of computational algorithms and data-structures
- Weakness
  - Models only one binary relationship

- 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

第十一章 空间网络模型与查询

- 11.1 Motivation, and use cases
- 11.2 Example spatial networks
- 11.3 Conceptual model
- 11.4 Need for SQL extensions
- 11.5 CONNECT statement
- 11.6 RECURSIVE statement
- 11.7 RECURSIVE in PostgreSQL
- 11.8 Storage and data structures
- 11.9 Algorithms for connectivity query
- 11.10 Algorithms for shortest path

Data Models of Spatial Networks

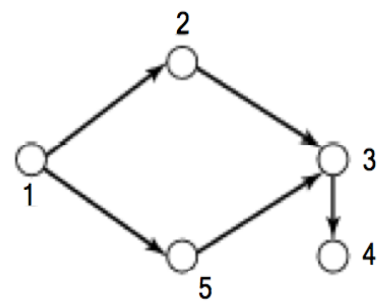
- Conceptual Model
  - Information Model: Entity Relationship Diagrams
  - Mathematical Model: Graphs
- Logical Data Model & Query Languages
  - Abstract Data types
  - Custom Statements in SQL
- Physical Data Model
  - Storage-Structures, File-Structures
  - Algorithms for common operations

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

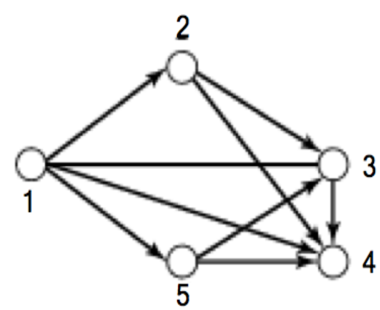


(a) Graph G

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

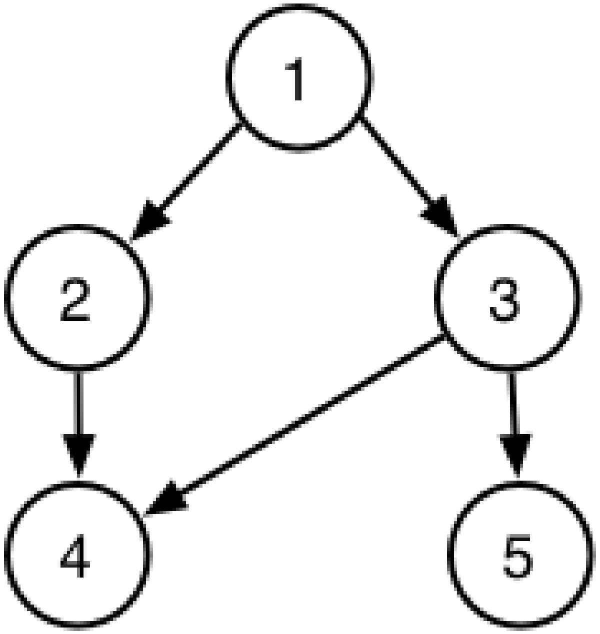
Supporting Graphs in SQL

- 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)



第十一章 空间网络模型与查询

- 11.1 Motivation, and use cases
- 11.2 Example spatial networks
- 11.3 Conceptual model
- 11.4 Need for SQL extensions
- 11.5 CONNECT statement
- 11.6 RECURSIVE statement
- 11.7 RECURSIVE in PostgreSQL
- 11.8 Storage and data structures
- 11.9 Algorithms for connectivity query
- 11.10 Algorithms for shortest path

Querying Graphs: Overview

- Relational Algebra
  - Can not express transitive closure queries

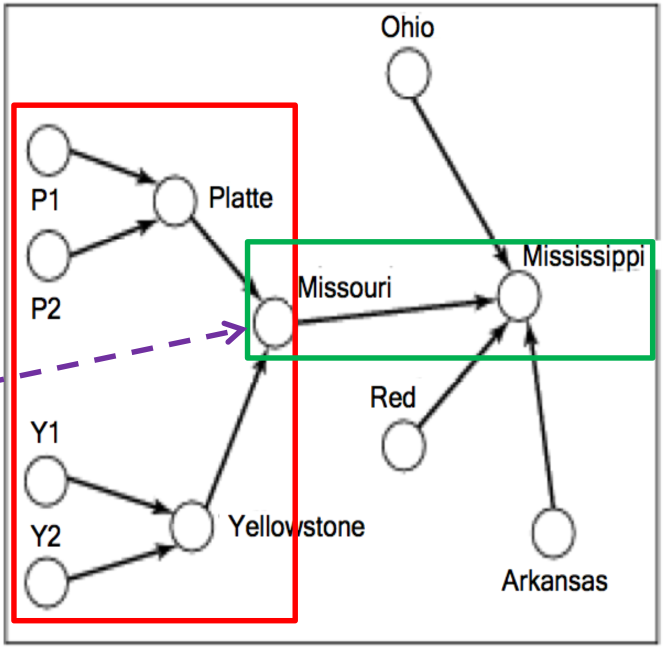
- Two ways to extend SQL to support graphs
  - Abstract Data Types
  - Custom Statements
    - SQL2 - CONNECT BY clause(s) in SELECT statement
    - SQL3 - WITH RECURSIVE statement
    - SQL3 - User defined data types

CONNECT BY: Input, Output

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

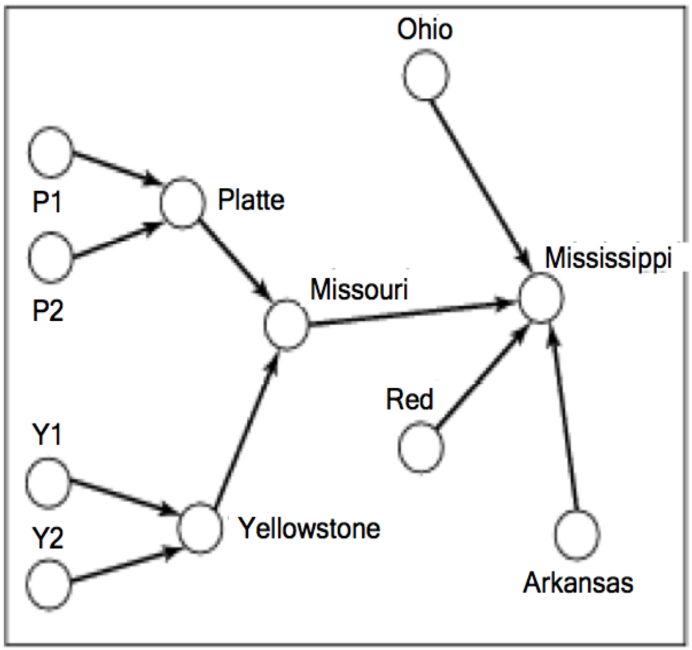
- Output: Transitive closure of G
  - Ex. Predecessors of S = Missouri
  - Ex. Successors of S = Missouri

(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	Dest
P1	Platte
P2	Platte
Y1	Yellowstone
Y2	Yellowstone
Platte	Missouri
Yellowstone	Missouri

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

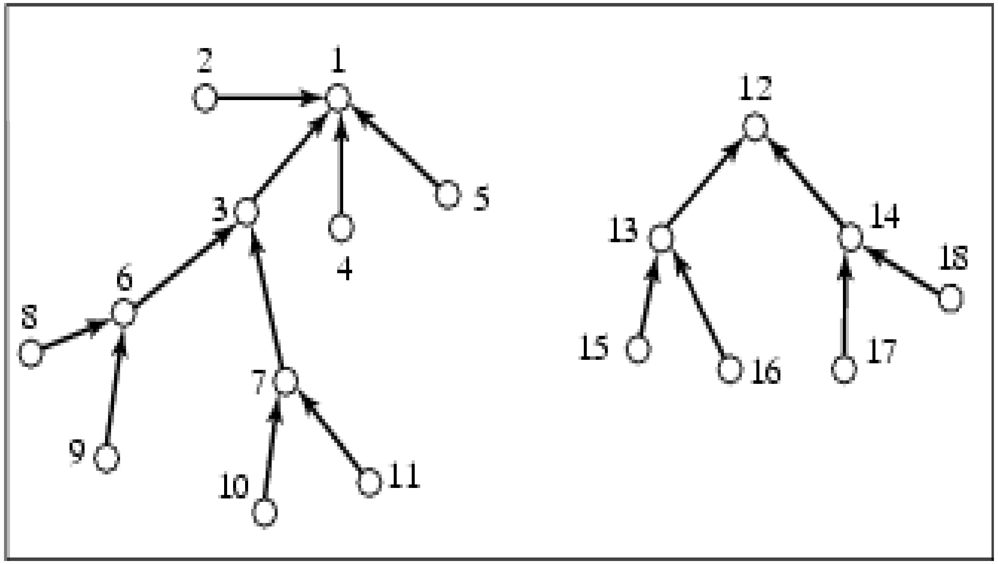
SQL CONNECT Exercise

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

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

- 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 source = **PRIOR** dest
START WITH dest = 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

第十一章 空间网络模型与查询

- 11.1 Motivation, and use cases
- 11.2 Example spatial networks
- 11.3 Conceptual model
- 11.4 Need for SQL extensions
- 11.5 CONNECT statement
- 11.6 RECURSIVE statement
- 11.7 RECURSIVE in PostgreSQL
- 11.8 Storage and data structures
- 11.9 Algorithms for connectivity query
- 11.10 Algorithms for shortest path

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

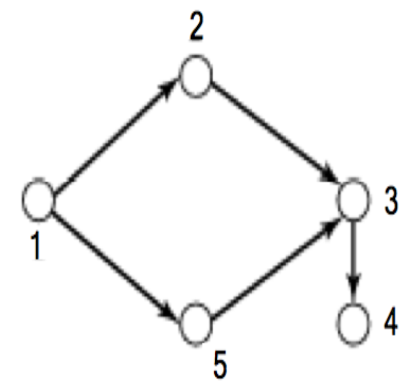
WITH RECURSIVE: Input, Output

- 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) — Description of Result Table

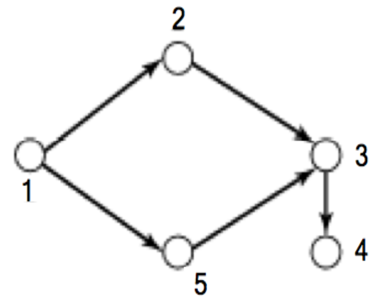
AS (**SELECT** source, dest **FROM** R) — Initialization Query

UNION

(**SELECT** R.source, X.dest
FROM R, X
WHERE R.dest = X.source) — 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 )
```

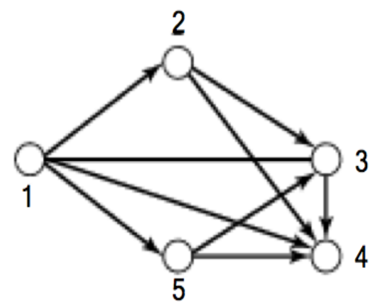


(a) Graph G

R

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

(b) Relation form



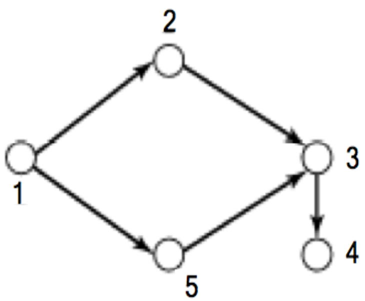
(c) Transitive closure (G) = Graph G

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

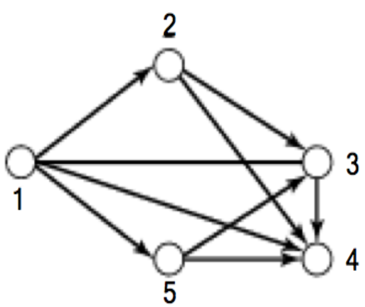


(a) Graph G

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

第十一章 空间网络模型与查询

- 11.1 Motivation, and use cases
- 11.2 Example spatial networks
- 11.3 Conceptual model
- 11.4 Need for SQL extensions
- 11.5 CONNECT statement
- 11.6 RECURSIVE statement
- 11.7 RECURSIVE in PostgreSQL
- 11.8 Storage and data structures
- 11.9 Algorithms for connectivity query
- 11.10 Algorithms for shortest path

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
1

Working
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
2

Query Q
1

Working W
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;
}
```

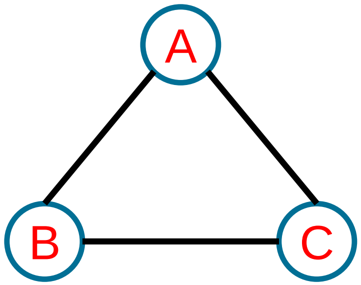
With Recursive Limitation

- 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，右图的关系未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

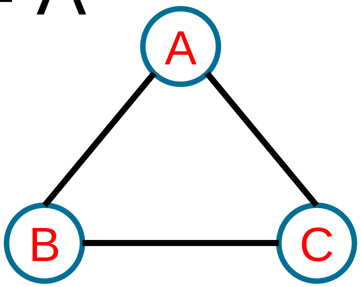
- with recursive X(node, depth) as (

    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)
- ……

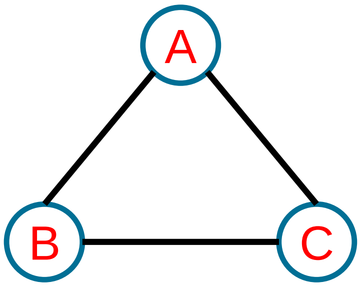
With Recursive Example 2

- 避免重复遍历节点path和cycle
- with recursive X(node, depth, path, circle) 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 circle)
```

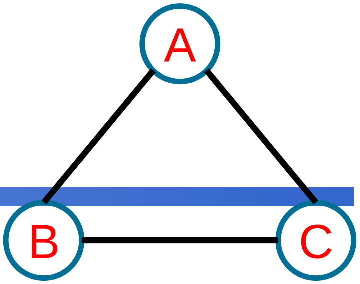
- PostgreSQL中的数组
  - 初始化：array[…]
  - 增加元素：path || end
  - 判断元素是否在数组中：end = any(path)

With Recursive Example 2



- with recursive X(node, depth, path, circle) 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 circle)
- 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;
```

- 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

第十一章 空间网络模型与查询

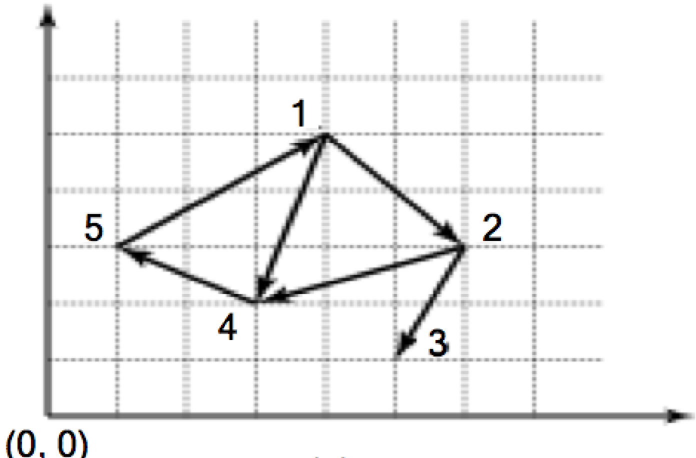
- 11.1 Motivation, and use cases
- 11.2 Example spatial networks
- 11.3 Conceptual model
- 11.4 Need for SQL extensions
- 11.5 CONNECT statement
- 11.6 RECURSIVE statement
- 11.7 RECURSIVE in PostgreSQL
- 11.8 Storage and data structures
- 11.9 Algorithms for connectivity query
- 11.10 Algorithms for shortest path

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

Main Memory Data-Structures

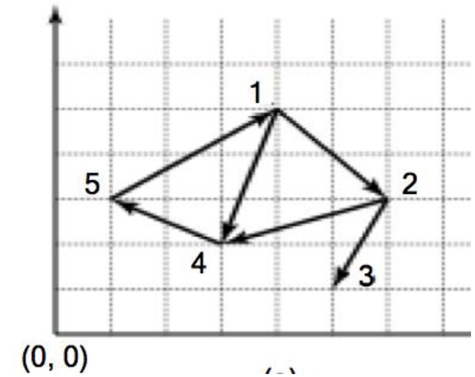
- Adjacency matrix
  - M[A, B] = 1 if and only if edge(vertex A, vertex B) exists
- Adjacency list :
  - Maps a vertex to a list of its successors



(a)

Disk-based Tables

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



Spatial Network Storage

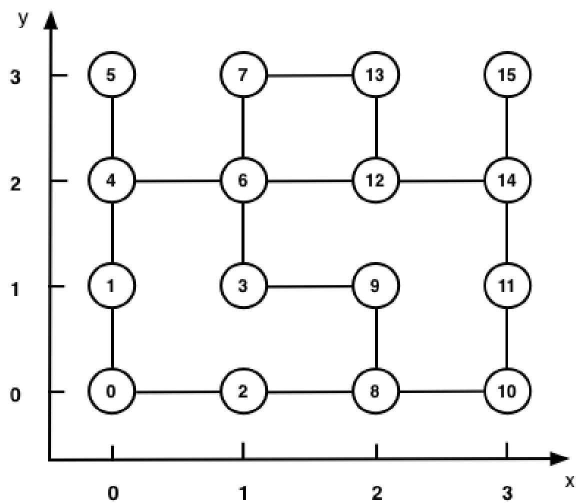
- Problem Statement
  - Given a spatial network
  - Find efficient data-structure to store it on disk sectors
  - Goal - Minimize I/O-cost of operations
    - Find(), Insert(), Delete(), Create()
    - Get-A-Successor(), Get-Successors()
  - Constraints
    - Spatial networks are much larger than main 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

- Insight:
  - 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

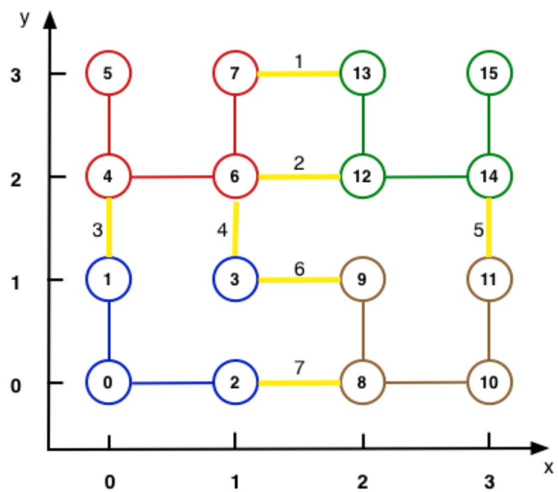
- Which partitioning reduces disk I/O for graph operations?
  - Choice 1: Geometric partition
  - Choice 2: Min-cut Graph Partition
  - Choice 2 cuts fewer edges and is preferred
  - Assuming uniform querying popularity across edges



Sample Network

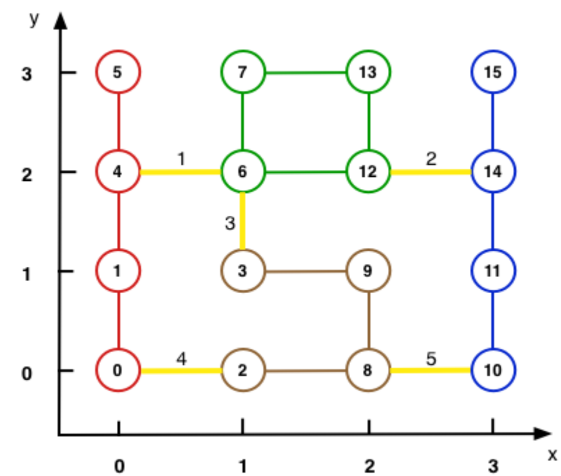
Node

Edge



Sample Network
Different data pages in different colors

Cut edge

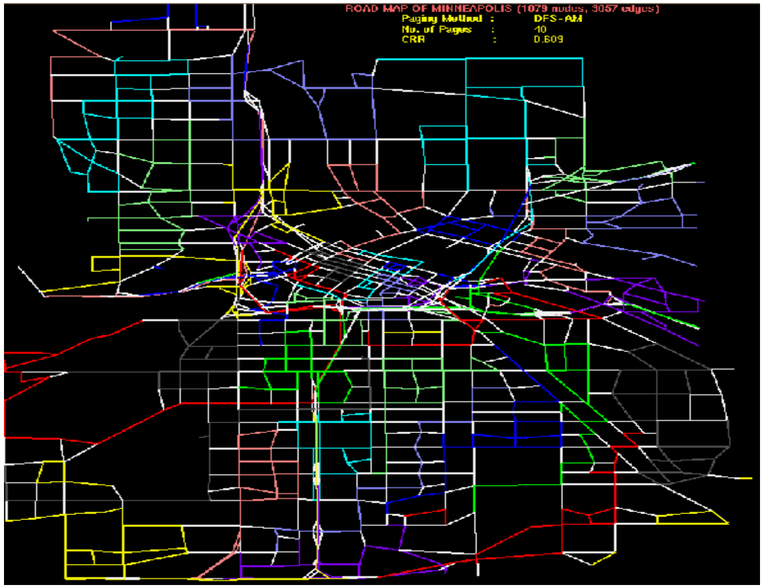


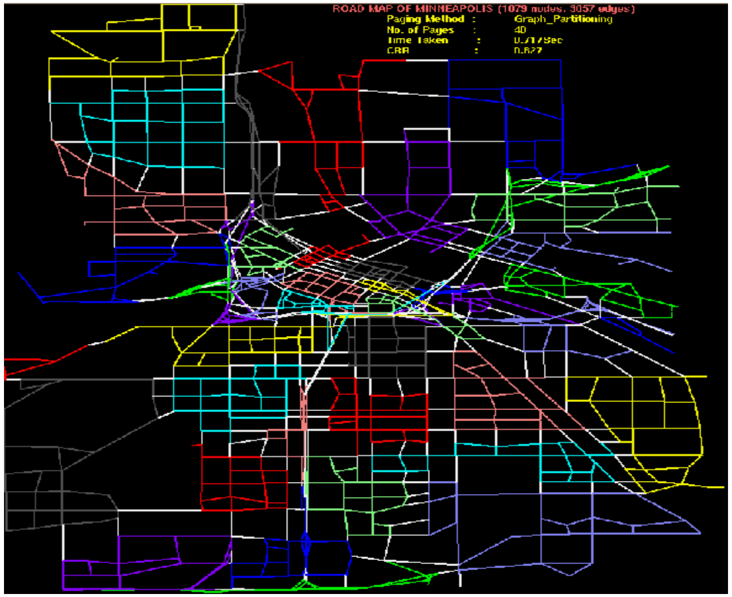
Sample Network
Different data pages in different colors

Cut edge

Graph Based Storage Methods

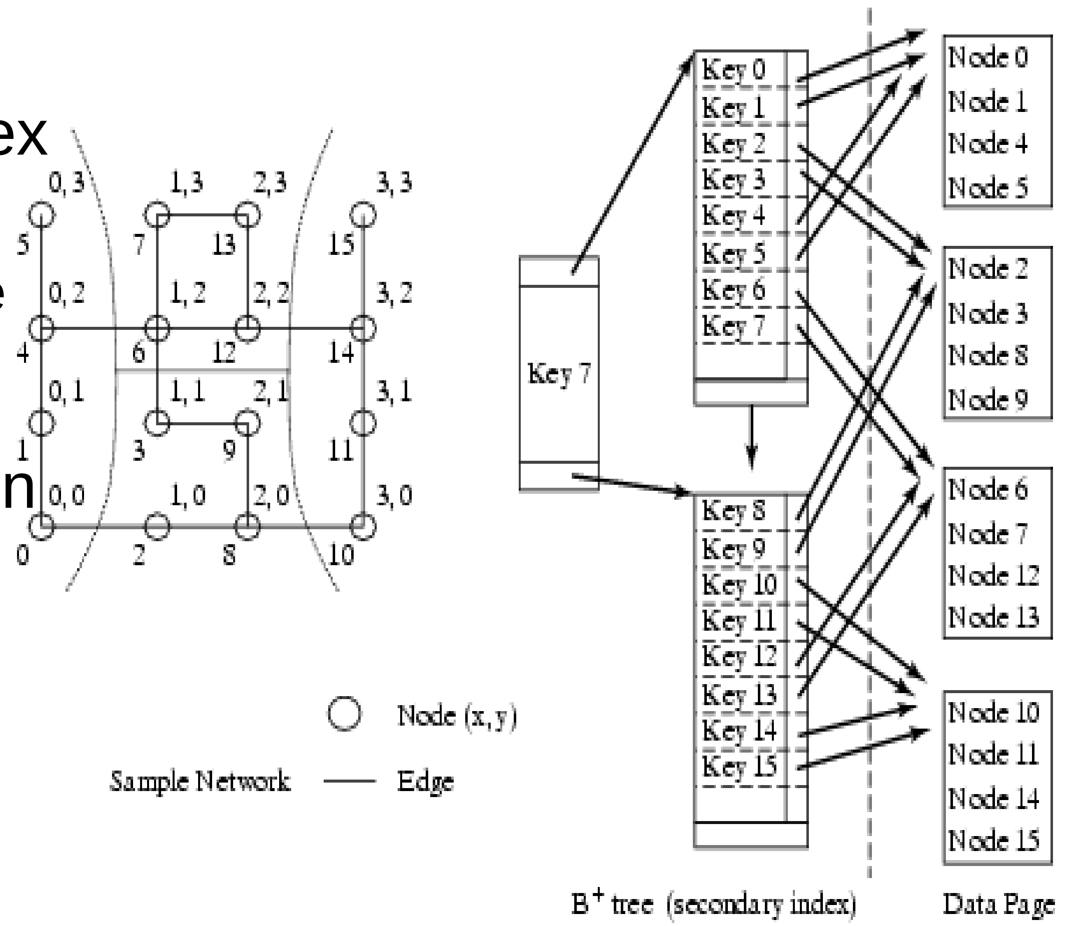
- Consider two disk-paging of Minneapolis (明尼阿波利斯) major roads
  - Non-white edges => node pair in same page
  - White edge are cut-edges
  - Node partitions on right has fewer cut-edges and is preferred => higher CRR





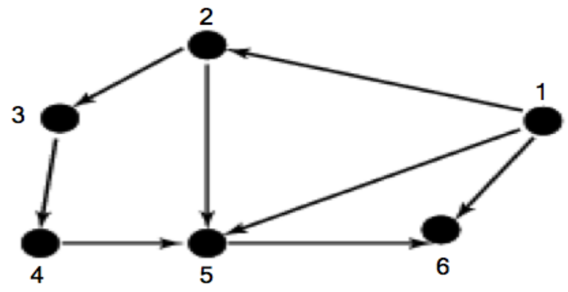
Clustering and Storing a Sample Network

- 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

第十一章 空间网络模型与查询

- 11.1 Motivation, and use cases
- 11.2 Example spatial networks
- 11.3 Conceptual model
- 11.4 Need for SQL extensions
- 11.5 CONNECT statement
- 11.6 RECURSIVE statement
- 11.7 RECURSIVE in PostgreSQL
- 11.8 Storage and data structures
- 11.9 Algorithms for connectivity query
- 11.10 Algorithms for shortest path

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

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

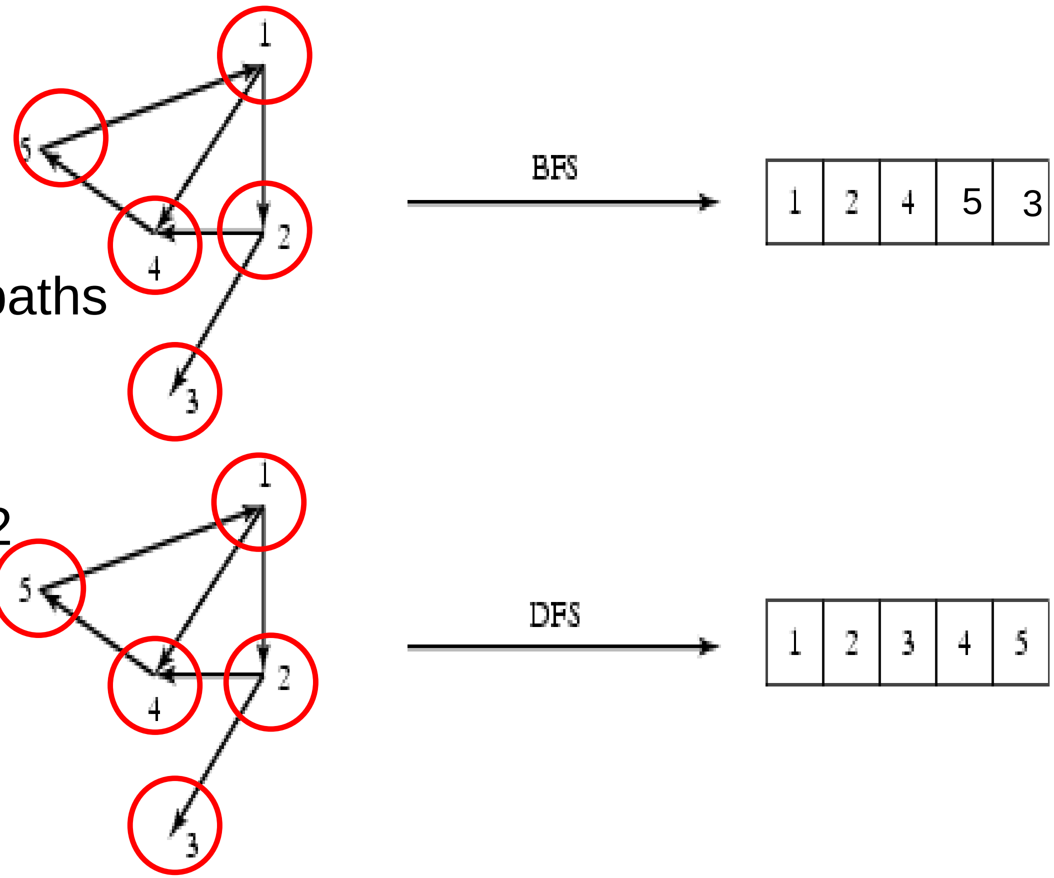
- Building blocks
  - Connectivity(A, B): Is node B reachable from node A?
  - Shortest path(A, B): Identify the least cost path from node A to node B

Algorithms

- Main memory
  - Connectivity: Breadth first search, Depth first search
  - Shortest path: Dijkstra's algorithm, A*
- Disk-based
  - Shortest path - Hierarchical routing algorithm
  - Connectivity strategies are in SQL3

Algorithms for Connectivity Query

- Breadth first search
  - Visit descendent by generation
  - Children before grandchildren
  - Example: 1 - (2,4) - (3, 5)
- Depth first search
  - Try a path till dead-end
  - Backtrack to try different paths
  - Like a maze game
  - Example: 1-2-3-2-4-5
  - Note backtrack from 3 to 2



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

第十一章 空间网络模型与查询

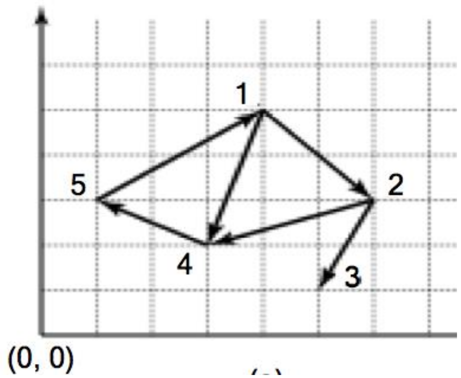
- 11.1 Motivation, and use cases
- 11.2 Example spatial networks
- 11.3 Conceptual model
- 11.4 Need for SQL extensions
- 11.5 CONNECT statement
- 11.6 RECURSIVE statement
- 11.7 RECURSIVE in PostgreSQL
- 11.8 Storage and data structures
- 11.9 Algorithms for connectivity query
- 11.10 Algorithms for shortest path

Shortest Path Algorithms

- Iterate
  - Expand most promising descent node
    - Dijkstra's: try closest descendent to self
    - A* : try closest descendent to both destination and self
  - Update current best path to each node, if a better path is found
- Till destination node is expanded

Dijkstra’s algorithm

- Dijkstra’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



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	$\sqrt{18}$

Dijkstra's algorithm

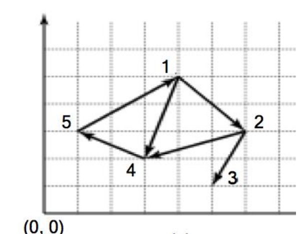
Node (R)

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	$\sqrt{18}$

- Example:
  - Consider shortest_path(1,5) for graph
  - Iteration 1 expands node 1 and edges (1,2), (1,4)
    - set cost(1,2) = sqrt(8); cost(1,4) = sqrt(10) using Edge table
  - Iteration 2 expands least cost node 2 and edges (2,3), (2,4)
    - set cost(1,3) = sqrt(8) + sqrt(5)
  - Iteration 3 expands least cost node 4 and edges (4,5)
    - set 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)

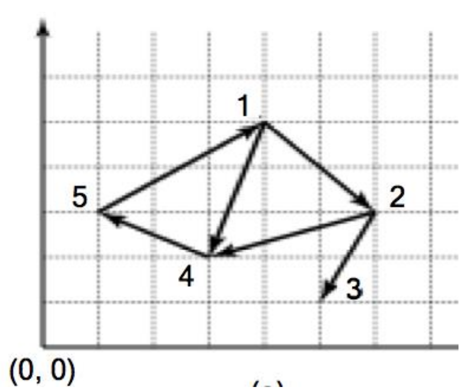


A*

- Best first algorithm
  - Similar to Dijkstra's algorithm with one change
  - Cost(node) = actual_cost(source, node) + estimated_cost(node, destination)
  - estimated_cost should be an underestimate of actual cost
    - Example - euclidean distance
- Given effective estimated_cost() function, it is faster than Dijkstra's algorithm

A*

- Example:
  - Revisit shortest_path(1,5) for graph in
  - Iteration 1 expands node 1 and edges (1,2), (1,4)
    - set actual_cost(1,2) = sqrt(8); actual_cost(1,4) = sqrt(10);
    - estimated_cost(2,5) = 5; estimated_cost(4,5) = sqrt(5)
  - Iteration 2 expands least cost node 4 and edges (4,5)
    - set actual_cost(1,5) = sqrt(10) + sqrt(5), estimated_cost(5,5) = 0
  - Iteration 3 expands node 5
  - Answer is the path (1-4-5)



Node (R)

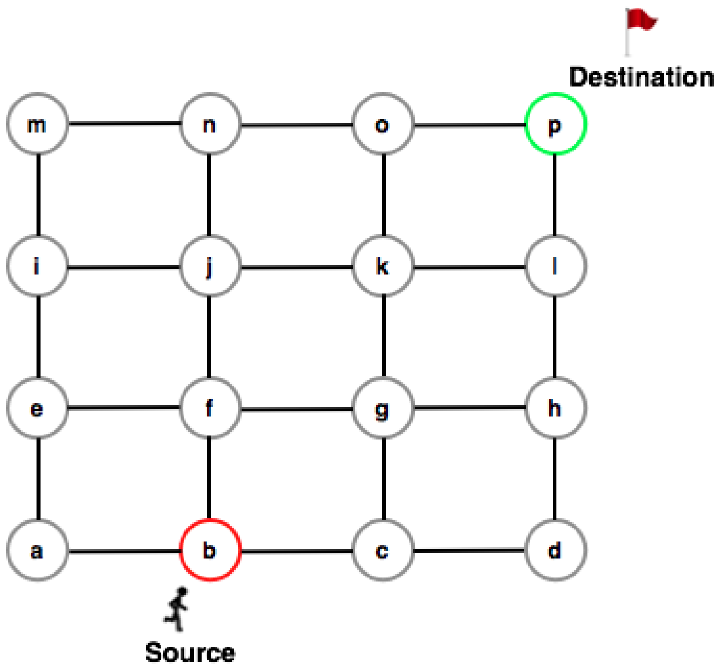
id	x	y
1	4.0	5.0
2	6.0	3.0
3	5.0	1.0
4	3.0	2.0
5	1.0	3.0

Edge (S)

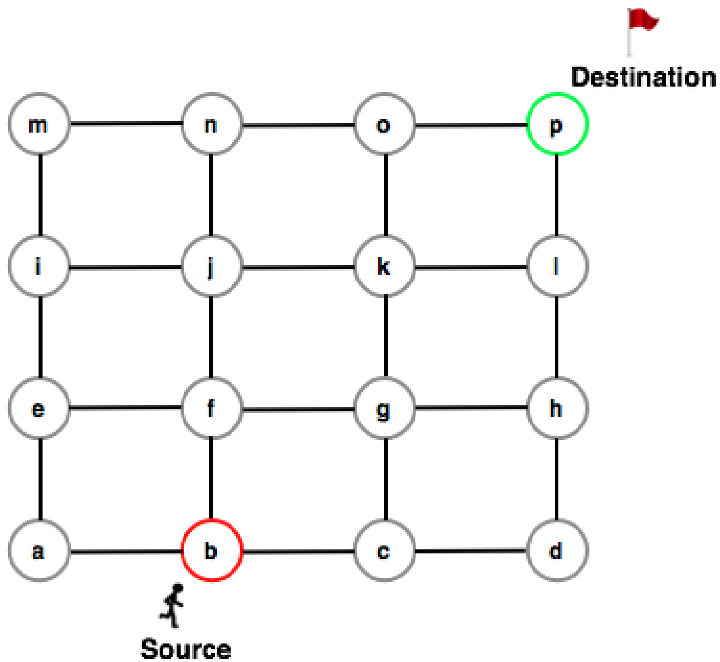
source	dest	distance
1	2	$\sqrt{8}$
1	4	$\sqrt{10}$
2	3	$\sqrt{5}$
2	4	$\sqrt{10}$
4	5	$\sqrt{5}$
5	1	$\sqrt{18}$

Dijkstra's vs. A*

Wave 0:



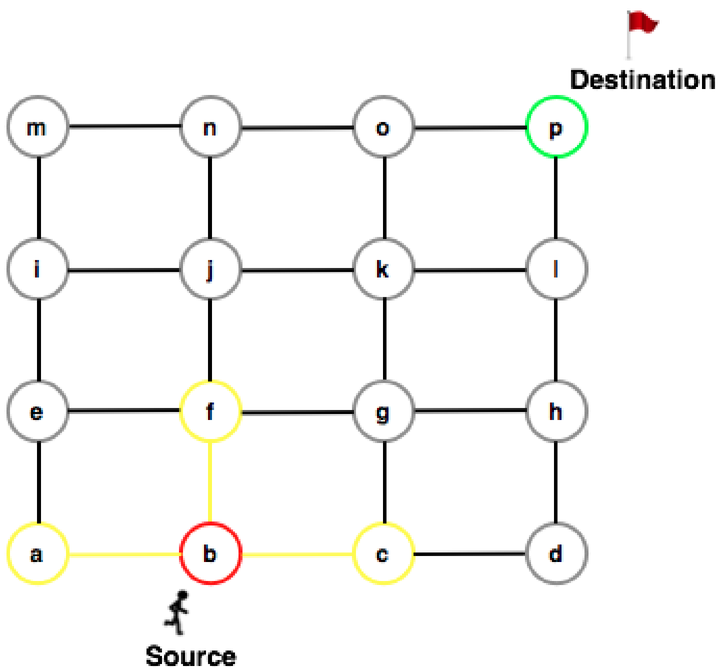
Dijkstra's Algorithm



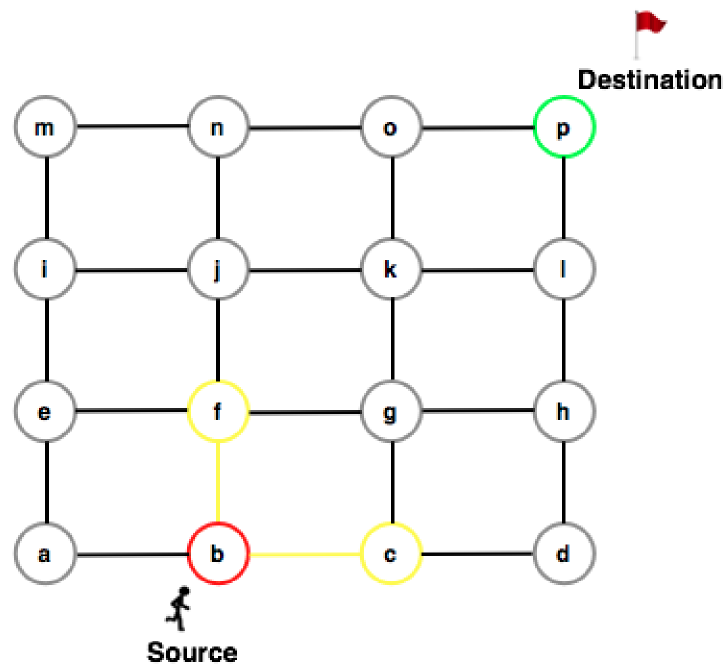
A* Algorithm

Dijkstra's vs. A*

Wave 1:



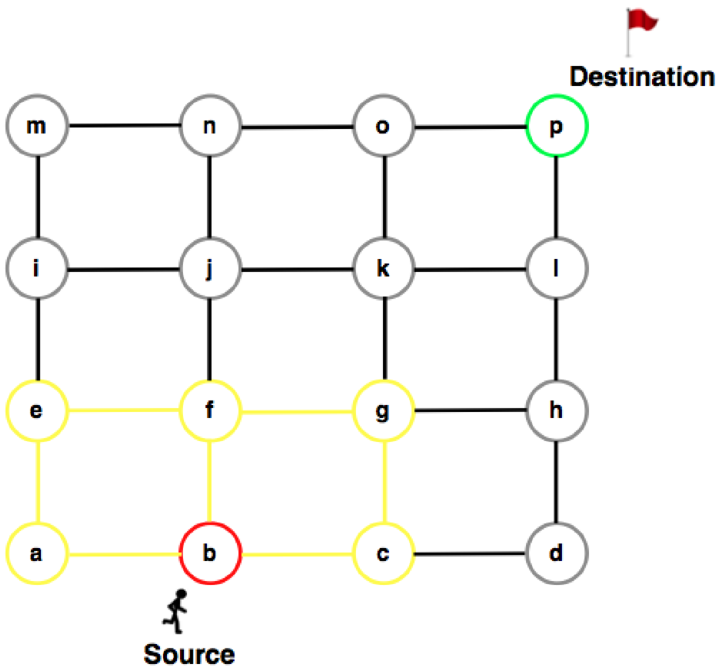
Dijkstra's Algorithm



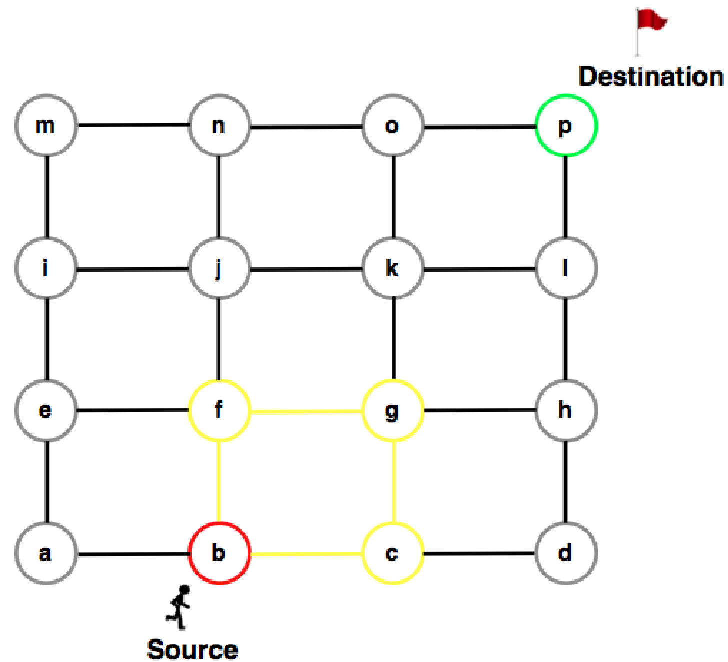
A* Algorithm

Dijkstra’s vs. A*

Wave 2:



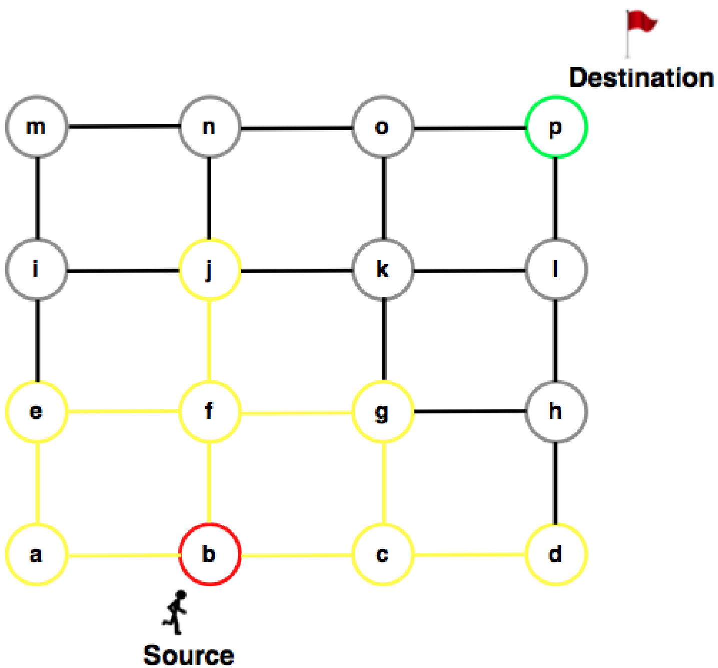
Dijkstra’s Algorithm



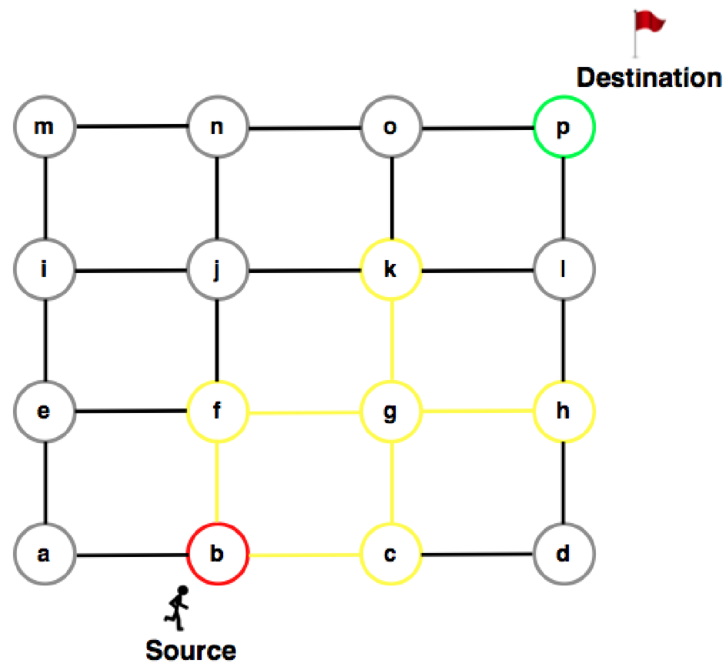
A* Algorithm

Dijkstra’s vs. A*

Wave 3:



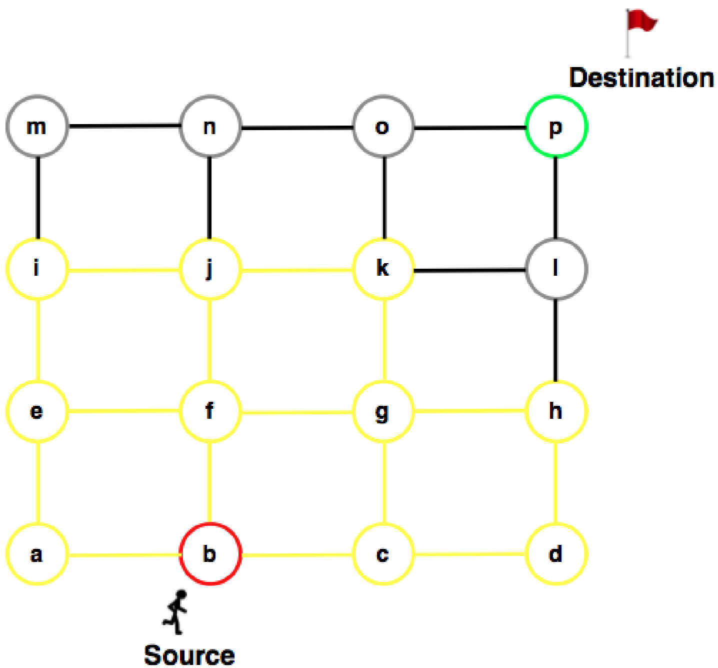
Dijkstra’s Algorithm



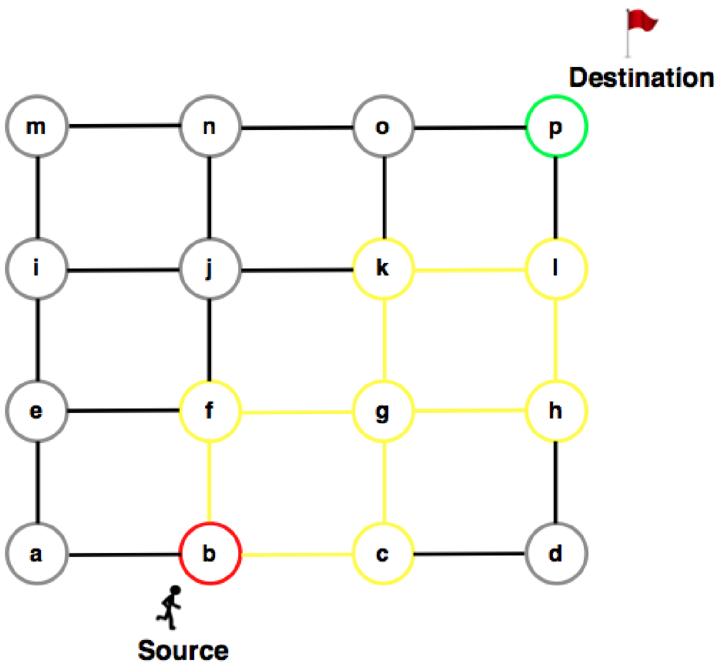
A* Algorithm

Dijkstra’s vs. A*

Wave 4:



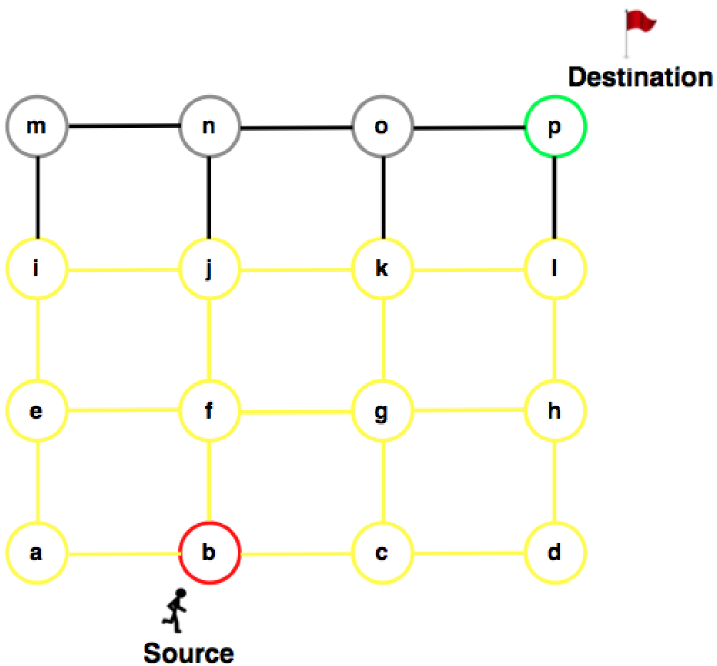
Dijkstra’s Algorithm



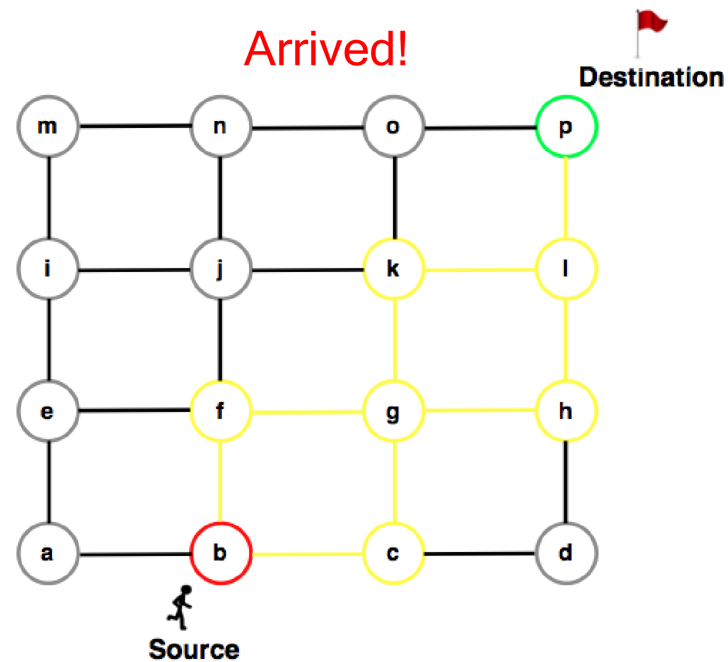
A* Algorithm

Dijkstra's vs. A*

Wave 5:



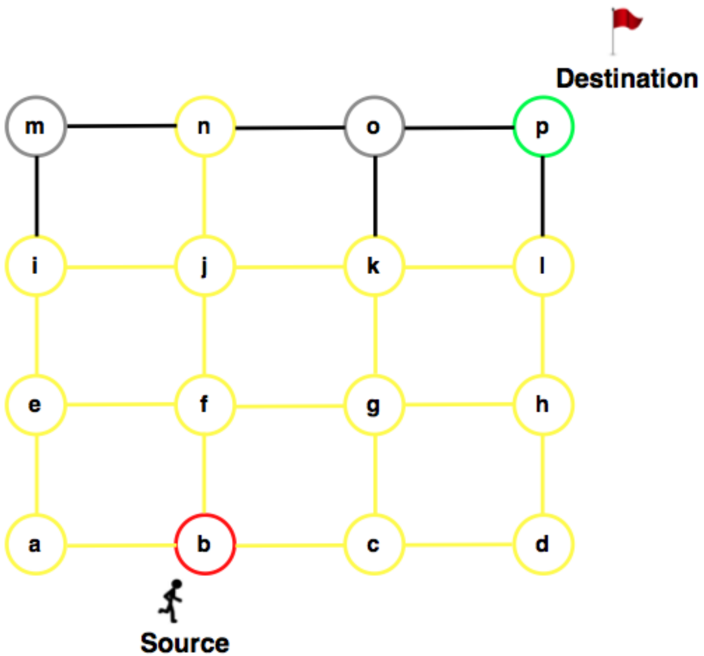
Dijkstra's Algorithm



A* Algorithm

Dijkstra’s vs. A*

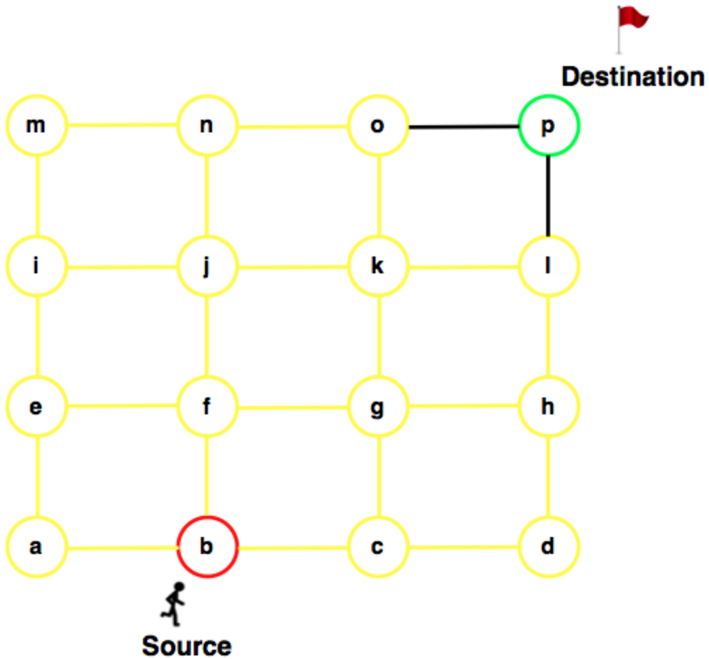
Wave 6:



Dijkstra’s Algorithm

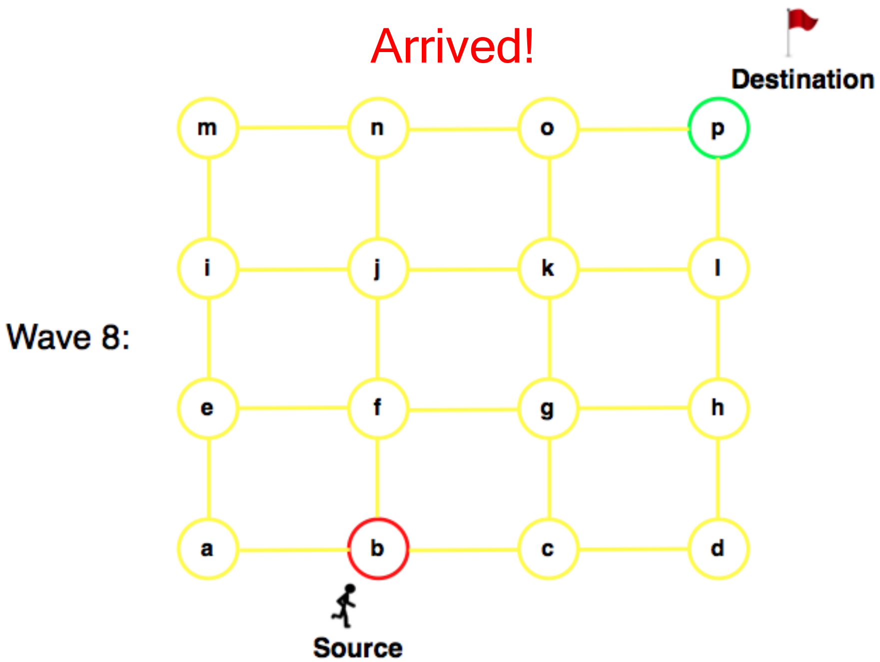
Dijkstra's vs. A*

Wave 7:



Dijkstra's Algorithm

Dijkstra's vs. A*



Dijkstra's Algorithm

Shortest Path Algorithms

- Iterate
  - Expand most promising node
    - Dijkstra's: try closest descendent to self
    - A* : try closest descendent to both destination and self
  - Update current best path to each node, if a better path is found
- Till destination node is expanded

- Correct assuming
  - Sub-path optimality
  - Fixed, positive and additive edge costs
  - A*: underestimate function

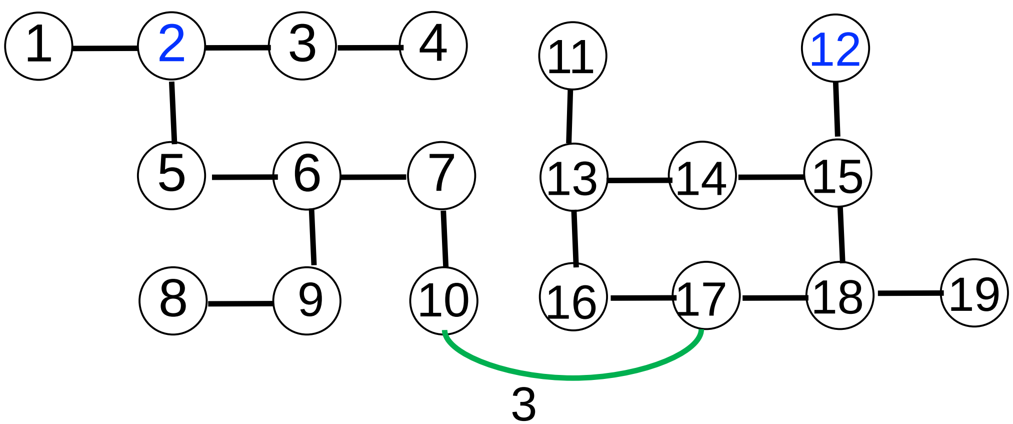
Shortest Path Strategies - 3

- 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 memories
  - 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

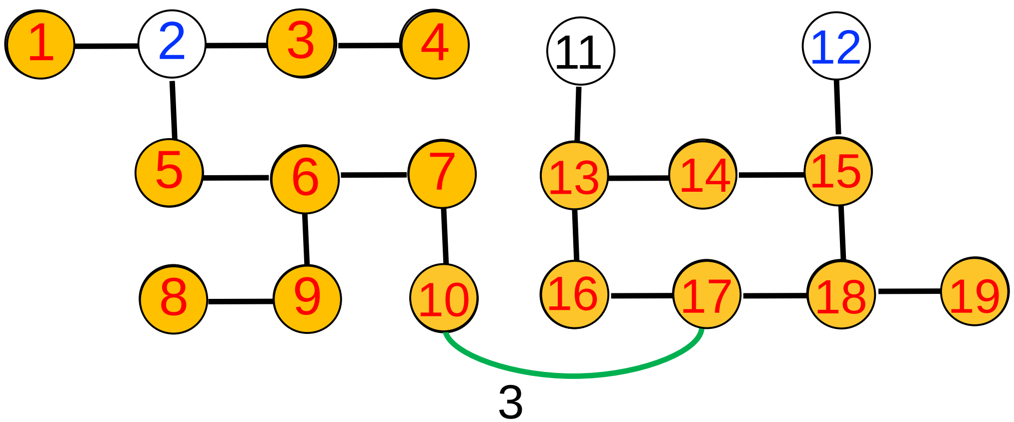


The edge length is 1 for every edge, except length(edge (11-18)) = 3

Trace Dijkstra's algorithm

- Tie-braking: prefer node with higher index

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	

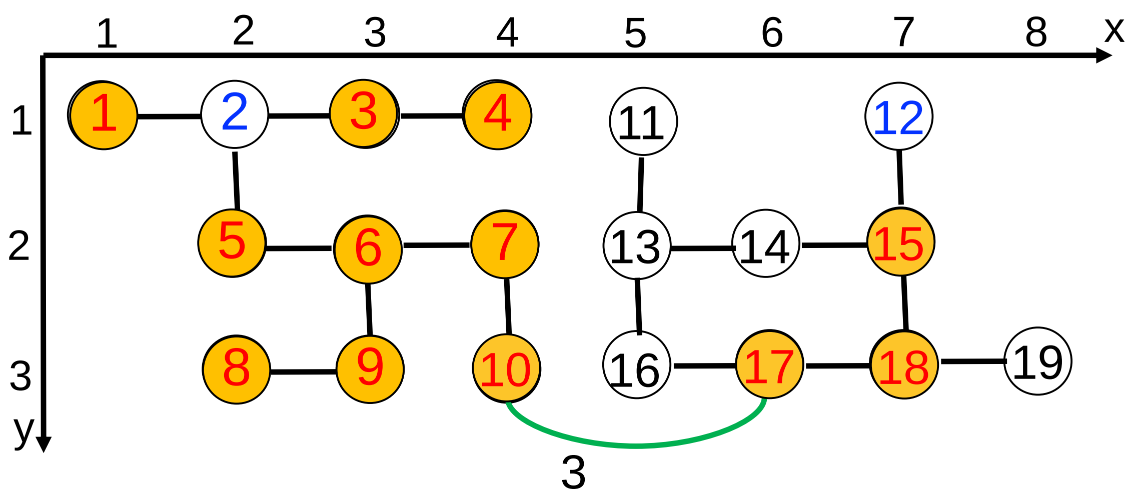
Q? Will A* (with Euclidean distance) be efficient?

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

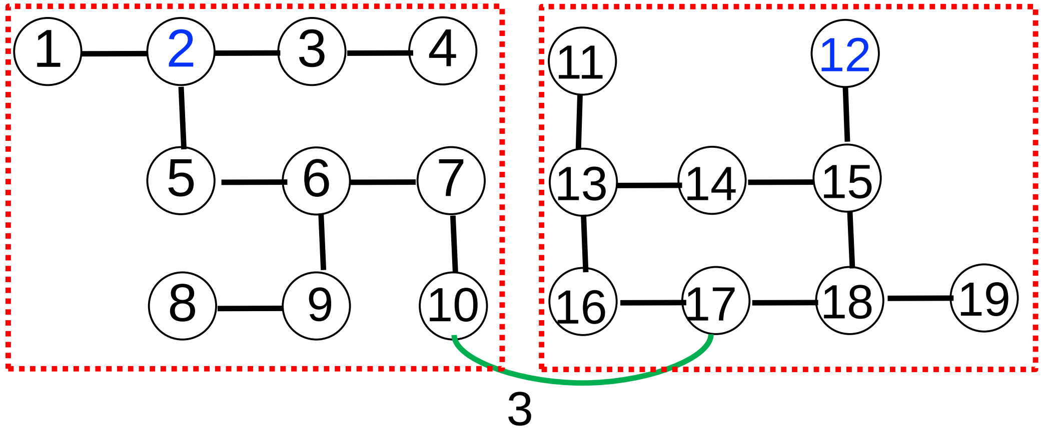


Expand:	Open List:
2	1, 3, 5
3	1, 4, 5
4	1, 5
5	1, 6
6	1, 7, 9
7	1, 9,10
10	1,9,17
1	9,17
9	8,17
17	8, 16,18
8	16,18
18	15,16,19
15	12,14,16,19
12	

Q? How may we reduce computation cost further?

Core Idea of the 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

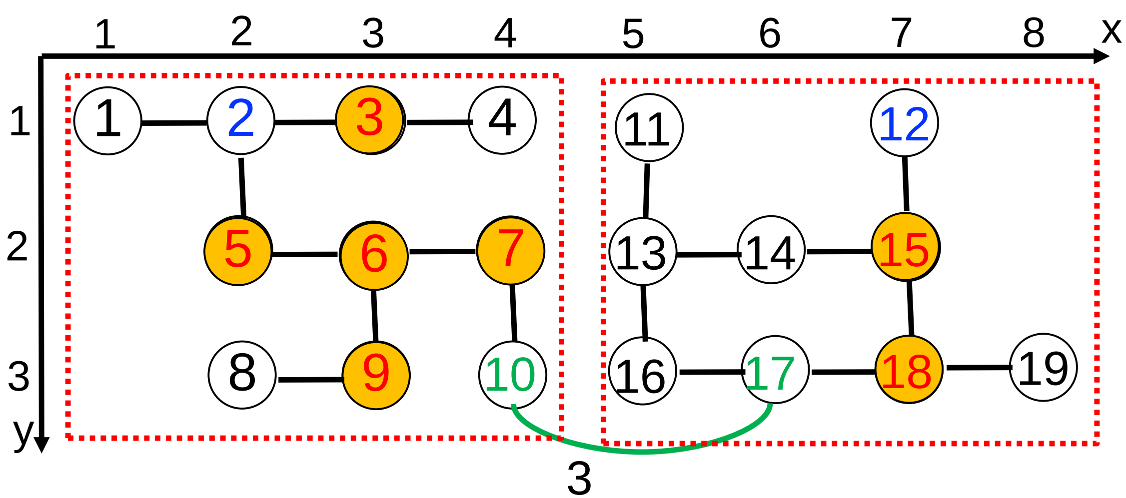
#Nodes expanded: 7 + 4 = 11

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	

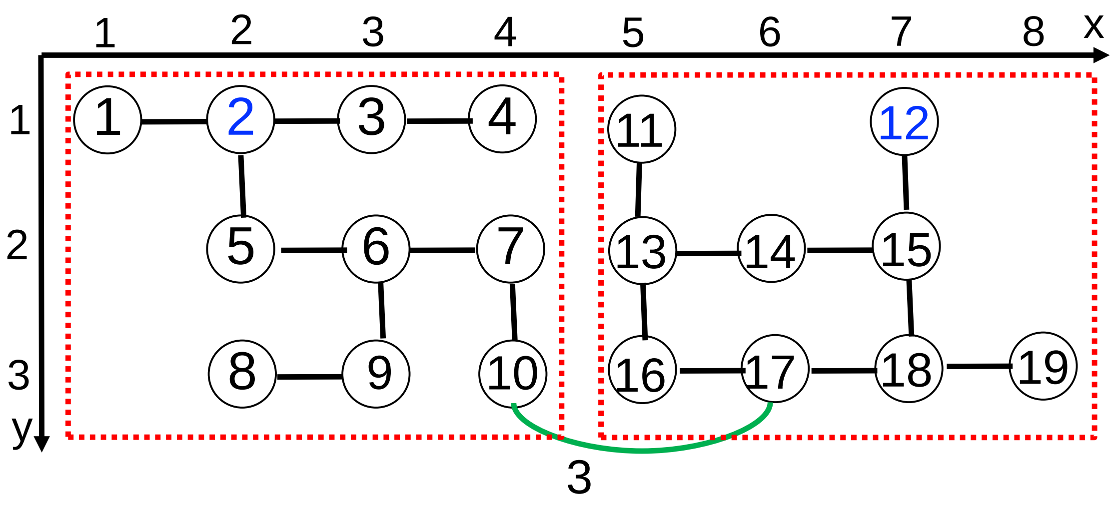
7 nodes expanded in SP(2 , 10) using A*

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?

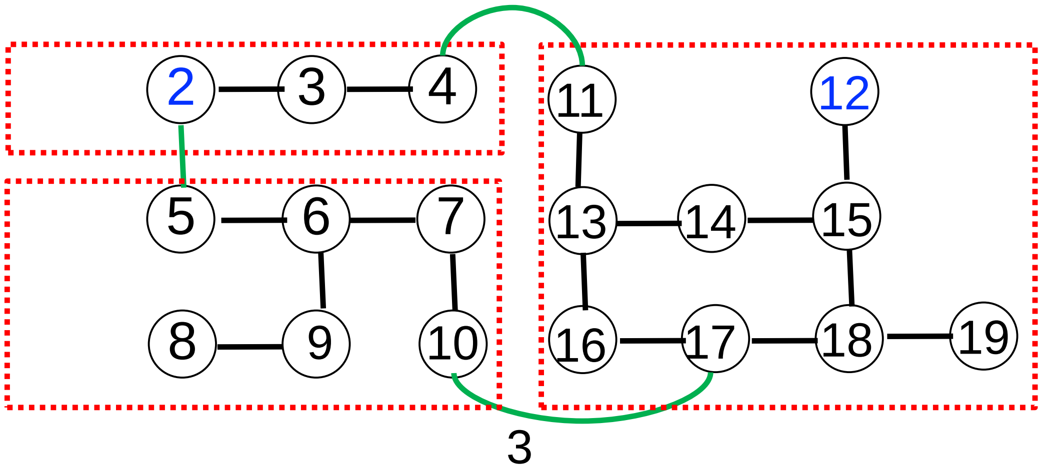
Algorithm (World View)	# of nodes expanded)
Dijkstra's (Graphs)	18
A* (Spatial Graphs)	14
Hierarchical Routing (Islands)	7 + 4 = 11



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

Challenges: Multiple Islands & Bridges

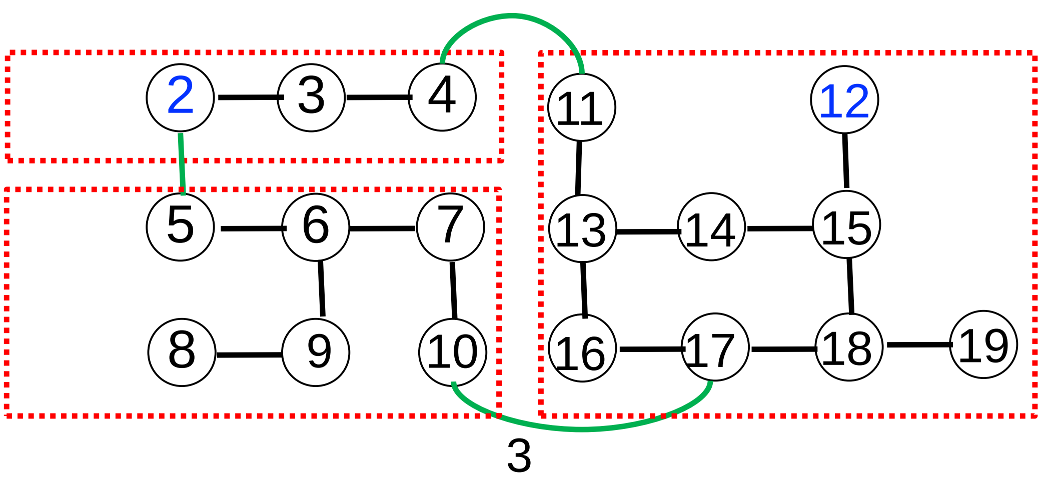
- Invariant: SP(2,12) must use one or more bridges
- Challenge 1: Multiple islands increase computational cost
- Challenge 2: Multiple bridges per island increase computational cost

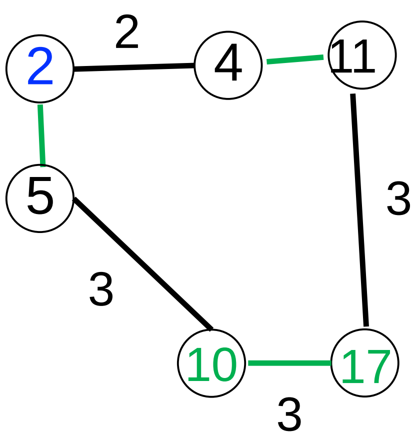


Hierarchical Algorithm with Multiple Islands

Data Structures:

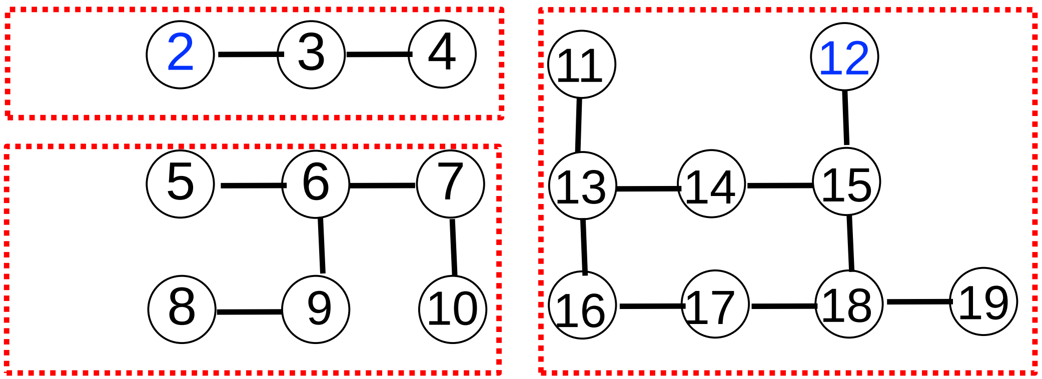
- Port node (a.k.a., boundary node) : a Node with edges to multiple islands
- Port Graph (a.k.a., boundary graph) :
- Island graphs (a.k.a., fragment graphs)
- Precompute & store
  - Shortest_path_costs: SPC(node i, node j) for all (or selected) node pairs (j,j)

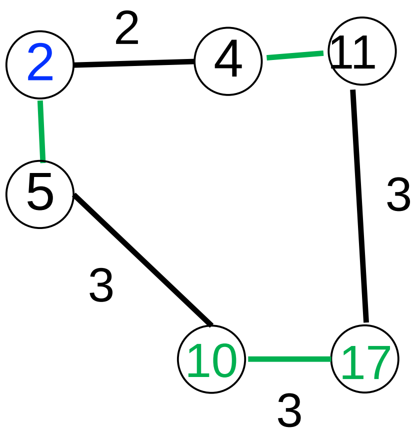




Hierarchical Algorithm with Multiple Islands

- 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





Shortest Path Strategies - 3 (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

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

Shortest Path Strategies – 3 (Illustration of the Algorithm)

- Figure 11.7(a) - fragments of source and destination nodes
- Figure 11.7(b) - computing summary of optimal path using
  - Boundary graph and 2 fragments
  - Note use of boundary edges only in the path computation
- Figure 11.8(a) - the summary of optimal path using boundary edges
- Figure 11.8(b) - expansion back to optimal path in original graph

Hierarchical Routing Algorithm-Step 1

- Step 1: Choose Boundary Node Pair
  - Minimize $COST(S,B_a)+COST(B_a,B_d)+COST(B_d,D)$
  - Determining Cost May Be Non-Trivial

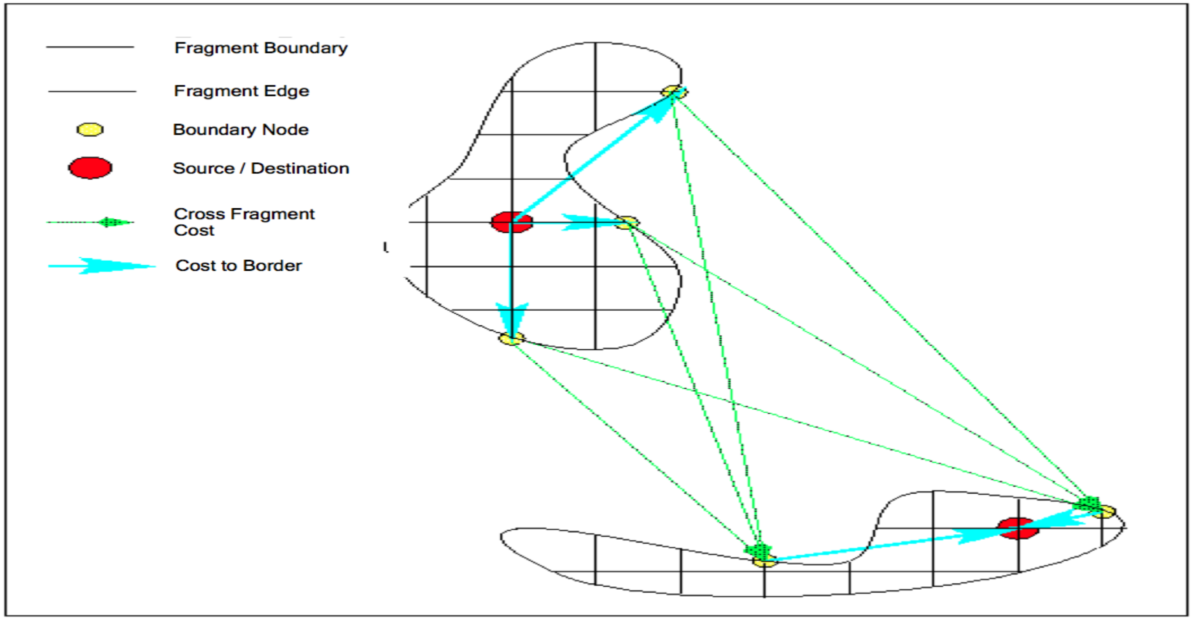


Fig 11.7(a)

Hierarchical Routing- Step 2

- Step 2: Examine Alternative Boundary Paths
  - Between Chosen Pair (B_a,B_d) of boundary nodes

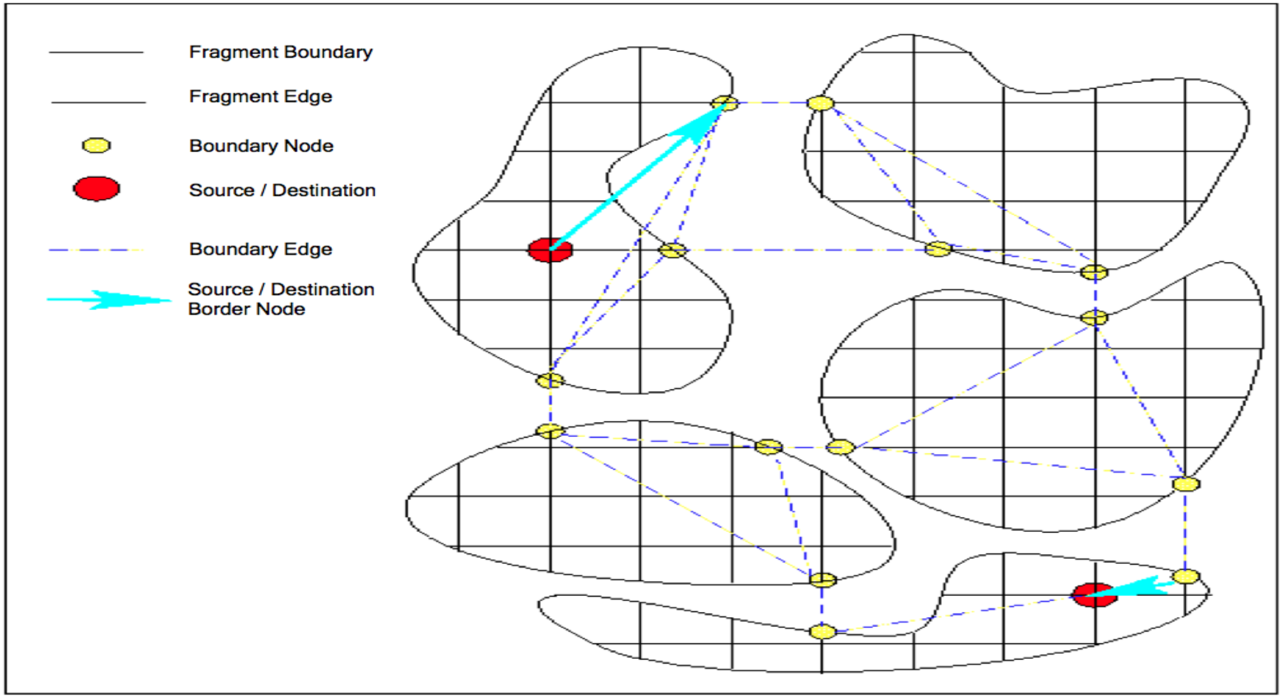
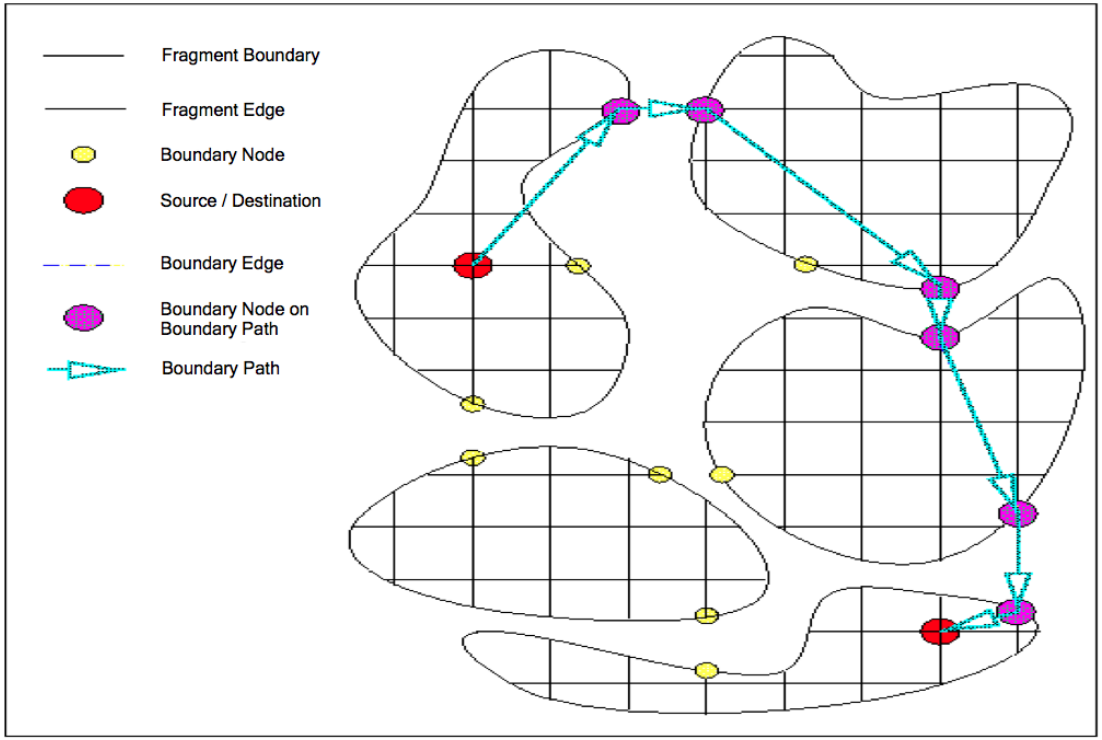


Fig 11.7(b)

Hierarchical Routing- Step 2 Result

- Step 2 Result: Shortest Boundary Path

Fig 11.8(a)



Hierarchical Routing- Step 3

- Step 3: Expand Boundary Path: $(B_{a1},B_d) \rightarrow B_{a1}\ B_{da2}\ B_{da3}\ B_{da4}\ldots B_d$
- Boundary Edge $(B_{ij},B_j) \rightarrow$ fragment path $(B_{i1},N_1N_2N_3\ldots\ldots.N_k,B_j)$

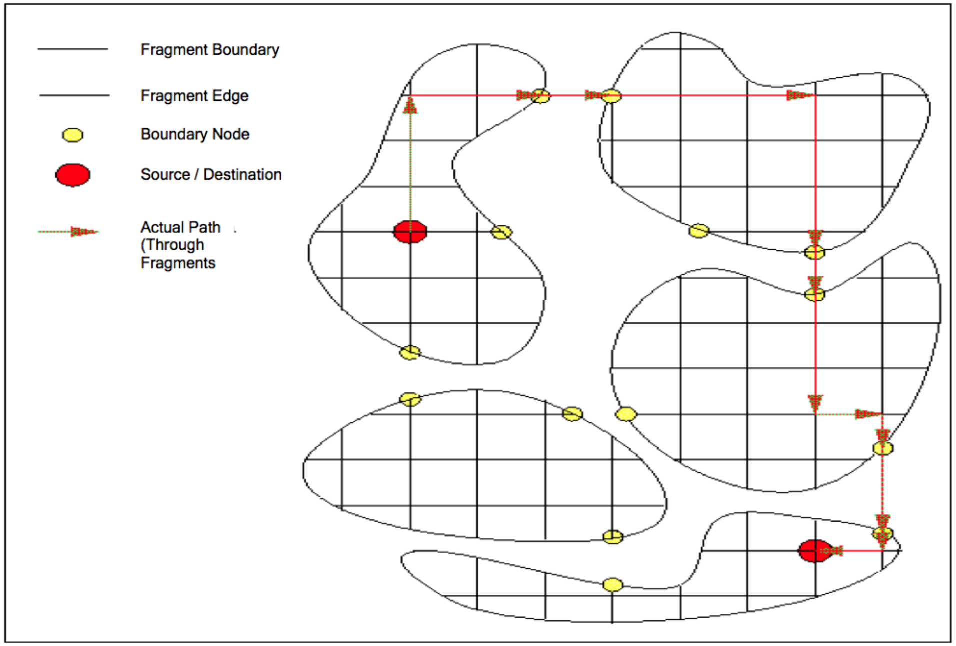


Fig 11.8(b)

pgRouting

- How to create a database and load pgrouting
  - createdb mydatabase
  - psql mydatabase -c “create extension postgis”
  - psql mydatabase -c “create extension pgrouting”

http://docs.pgrouting.org/latest/en/index.html

pgRouting Routing Functions

- pgr_apspJohnson
  - All Pairs Shortest Path, Johnson’s Algorithm
- pgr_apspWarshall
  - All Pairs Shortest Path, Floyd-Warshall Algorithm
- pgr_astar
  - Shortest Path A*
- pgr_bdAstar
  - Bi-directional A* Shortest Path

pgRouting Routing Functions

- pgr_bdDijkstra
  - Bi-directional Dijkstra Shortest Path
- pgr_dijkstra
  - Shortest Path Dijkstra
- pgr_driving_distance
  - Driving Distance
- pgr_kDijkstra
  - Mutliple destination Shortest Path Dijkstra

pgRouting Routing Functions

- pgr_ksp
  - K-Shortest Path
- pgr_trsp
  - Turn Restriction Shortest Path (TRSP)
- pgr_tsp
  - Traveling Sales Person

pgRouting Example

- 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
);
```

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

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
- Graph Query Processing
  - Building blocks - connectivity, shortest paths
  - Strategies - Best first, Dijkstra’s and Hierarchical routing
- Storage and access methods
  - Minimize CRR