CSE544: Principles of Database Systems

Datalog
Announcements

• Review 7 (Datalog) due next Monday

• Project Milestone due next Friday

• HW3 is posted, due the following Monday
Datalog

Review the following basic concepts from Lecture 5:

- Fact
- Rule
- Head and body of a rule
- Existential variable
- Head variable
Simple datalog programs

R encodes a graph

R =

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

T(x, y) :- R(x, y)
T(x, y) :- R(x, z), T(z, y)

What does it compute?
Simple datalog programs

R encodes a graph

\[
\begin{align*}
R &= T(x, y) \leftarrow R(x, y) \\
T(x, y) &\leftarrow R(x, z), T(z, y)
\end{align*}
\]

What does it compute?

Initially:
T is empty.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
Simple datalog programs

R encodes a graph

T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)

What does it compute?

Initially:
T is empty.

First iteration:
T =

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Simple datalog programs

R encodes a graph

T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)

What does it compute?

Initially:
T is empty.

First iteration:
T =

Second iteration:
T =

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Simple datalog programs

R encodes a graph

T(x,y) :- R(x,y)
T(x,y) :- R(x,z), T(z,y)

What does it compute?

Initially:
T is empty.

First iteration:
T =

Second iteration:
T =

Third iteration:
T =
Simple datalog programs

R encodes a graph

\[ T(x,y) :- R(x,y) \]
\[ T(x,y) :- R(x,z), T(z,y) \]

Initial iteration:
\[
\begin{array}{cc}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\end{array}
\]

T is empty.

Second iteration:
\[
\begin{array}{cc}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\end{array}
\]
\[
\begin{array}{cc}
1 & 1 \\
2 & 2 \\
1 & 3 \\
2 & 4 \\
1 & 5 \\
3 & 5 \\
\end{array}
\]

Third iteration:
\[
\begin{array}{cc}
1 & 2 \\
2 & 1 \\
2 & 3 \\
1 & 4 \\
3 & 4 \\
4 & 5 \\
\end{array}
\]
\[
\begin{array}{cc}
1 & 1 \\
2 & 2 \\
1 & 3 \\
2 & 4 \\
1 & 5 \\
3 & 5 \\
\end{array}
\]

Discovered 3 times!
Discovered twice

What does it compute?
Simple datalog programs

R encodes a graph

R=  

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Alternative ways to compute TC:

- Right linear
  \[
  T(x,y) :- R(x,y)
  \]
  \[
  T(x,y) :- R(x,z), T(z,y)
  \]

- Left linear
  \[
  T(x,y) :- R(x,y)
  \]
  \[
  T(x,y) :- T(x,z), R(z,y)
  \]

- Non-linear
  \[
  T(x,y) :- R(x,y)
  \]
  \[
  T(x,y) :- T(x,z), T(z,y)
  \]

Discuss pros/cons in class
Simple datalog programs

R encodes a colored graph

Compute TC (ignoring color):

Compute pairs of nodes connected by the same color (e.g. (2,4))

R =

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Blue</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Green</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Blue</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Red</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Yellow</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Simple datalog programs

R encodes a colored graph

\[
R = \begin{array}{|c|c|c|}
\hline
1 & Red & 2 \\
2 & Blue & 1 \\
2 & Green & 3 \\
1 & Blue & 4 \\
3 & Red & 4 \\
4 & Yellow & 5 \\
\hline
\end{array}
\]

Compute TC (ignoring color):

- \( T(x,y) :- R(x,c,y) \)
- \( T(x,y) :- R(x,c,z), T(z,y) \)

Compute pairs of nodes connected by the same color (e.g. (2,4))
Simple datalog programs

R encodes a colored graph

1
2
3
4
5

R=

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Red</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Blue</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Green</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>Blue</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Red</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>Yellow</td>
<td>5</td>
</tr>
</tbody>
</table>

Compute TC (ignoring color):

T(x,y) :- R(x,c,y)
T(x,y) :- R(x,c,z), T(z,y)

Compute pairs of nodes connected by the same color (e.g. (2,4))

T(x,c,y) :- R(x,c,y)
T(x,c,y) :- R(x,c,z), T(z,c,y)

Answer(x,y) :- T(x,c,y)
Simple datalog programs

R, G, B encodes a 3-colored graph

What does this program compute in general?

```
S(x,y) :- B(x,y)
S(x,y) :- T(x,z), B(z,y)
T(x,y) :- S(x,z), R(z,y)
T(x,y) :- S(x,z), G(z,y)
Answer(x,y) :- T(x,y)
```

CSE544 - Spring, 2013
Simple datalog programs

R, G, B encodes a 3-colored graph

What does this program compute in general?

S(x,y) :- B(x,y)
S(x,y) :- T(x,z),B(z,y)
T(x,y) :- S(x,z),R(z,y)
T(x,y) :- S(x,z),G(z,y)
Answer(x,y) :- T(x,y)

Answer: it computes pairs of nodes connected by a path spelling out these regular expressions:

• $S = (B.(R \text{ or } G))^*.B$
• $T = (B.(R \text{ or } G))^+$
Syntax of Datalog Programs

The schema consists of two sets of relations:

- **Extensional Database (EDB):** $R_1, R_2, \ldots$
- **Intentional Database (IDB):** $P_1, P_2, \ldots$

A datalog program $P$ has the form:

$$P: \quad P_{i1}(x_{11}, x_{12}, \ldots) :\text{-} body_1$$
$$\quad P_{i2}(x_{21}, x_{22}, \ldots) :\text{-} body_2$$
$$\quad \ldots$$

- Each head predicate $P_i$ is an IDB
- Each body is a conjunction of IDB and/or EDB predicates
- See lecture 2

Note: no negation (yet)! Recursion OK.
Naïve Datalog Evaluation Algorithm

Datalog program:

\[ P_{i1} :- \text{body}_1 \]
\[ P_{i2} :- \text{body}_2 \]
....
Naïve Datalog Evaluation Algorithm

Datalog program:

\[
P_{i1} :- \text{body}_1 \\
P_{i2} :- \text{body}_2 \\
\vdots
\]

\[
P_1 :- \text{body}_{11} \cup \text{body}_{12} \cup \ldots \\
P_2 :- \text{body}_{21} \cup \text{body}_{22} \cup \ldots \\
\vdots
\]

Group by IDB predicate
Naïve Datalog Evaluation Algorithm

Datalog program:

$$\begin{align*}
P_{i1} & :- \text{body}_1 \\
P_{i2} & :- \text{body}_2 \\
& \vdots
\end{align*}$$

$$\begin{align*}
P_1 & :- \text{body}_{11} \cup \text{body}_{12} \cup \ldots \\
P_2 & :- \text{body}_{21} \cup \text{body}_{22} \cup \ldots \\
& \vdots
\end{align*}$$

Each rule is a Select-Project-Join-Union query

Group by IDB predicate
Naïve Datalog Evaluation Algorithm

Datalog program:

\[ P_{i1} : - \text{body}_{1} \]
\[ P_{i2} : - \text{body}_{2} \]

\[ P_{1} : - \text{body}_{11} \cup \text{body}_{12} \cup \ldots \]
\[ P_{2} : - \text{body}_{21} \cup \text{body}_{22} \cup \ldots \]

Group by IDB predicate

Each rule is a Select-Project-Join-Union query.

Example:

\[ T(x,y) : - R(x,y) \]
\[ T(x,y) : - R(x,z), T(z,y) \]

?
Naïve Datalog Evaluation Algorithm

Datalog program:

\[ P_{i1} : - \text{body}_1 \]
\[ P_{i2} : - \text{body}_2 \]

\[ \vdots \]

Group by IDB predicate

\[ P_1 : - \text{body}_{11} \cup \text{body}_{12} \cup \ldots \]
\[ P_2 : - \text{body}_{21} \cup \text{body}_{22} \cup \ldots \]

\[ \vdots \]

Each rule is a Select-Project-Join-Union query

Example:

\[ T(x,y) : - R(x,y) \]
\[ T(x,y) : - R(x,z), T(z,y) \]

\[ \Rightarrow \]
\[ T(x,y) : - R(x,y) \cup \Pi_{xy}(R(x,z) \bowtie T(z,y)) \]
Naïve Datalog Evaluation Algorithm

Datalog program:

\[
P_{i1} :- \text{body}_1 \\
P_{i2} :- \text{body}_2 \\
... \\
\Rightarrow \\
P_1 :- \text{body}_{11} \cup \text{body}_{12} \cup ... \\
P_2 :- \text{body}_{21} \cup \text{body}_{22} \cup ... \\
... \\
\Rightarrow \\
P_1 :- \text{SPJU}_1 \\
P_2 :- \text{SPJU}_2 \\
... \\
\]

Naïve datalog evaluation algorithm:

\[
P_1 = P_2 = ... = \emptyset \\
\text{Loop} \\
\text{NewP}_1 = \text{SPJU}_1; \text{NewP}_2 = \text{SPJU}_2; ... \\
\text{if} (\text{NewP}_1 = P_1 \text{ and NewP}_2 = P_2 \text{ and } ...) \\
\text{then exit} \\
P_1 = \text{NewP}_1; P_2 = \text{NewP}_2; ... \\
\text{Endloop} \\
\]

Example:

\[
T(x,y) :- \text{R}(x,y) \\
T(x,y) :- \text{R}(x,z), T(z,y) \\
\Rightarrow \\
T(x,y) :- \text{R}(x,y) \cup \Pi_{xy}(\text{R}(x,z) \bowtie T(z,y)) \\
\]
Naïve Datalog Evaluation Algorithm

Datalog program:

\[ P_{i1} \leftarrow \text{body}_{11} \]
\[ P_{i2} \leftarrow \text{body}_{21} \]
\[ \quad \text{...} \]

\[ P_1 \leftarrow \text{body}_{11} \cup \text{body}_{12} \cup \text{...} \]
\[ P_2 \leftarrow \text{body}_{21} \cup \text{body}_{22} \cup \text{...} \]

\[ \quad \text{...} \]

Group by IDB predicate

Each rule is a Select-Project-Join-Union query

Naïve datalog evaluation algorithm:

\[ P_1 = P_2 = \ldots = \emptyset \]
Loop

New\(P_1 = \text{SPJU}_1;\) New\(P_2 = \text{SPJU}_2;\) \ldots
if (New\(P_1 = P_1 \) and New\(P_2 = P_2 \) and \ldots)
then exit
\[ P_1 = \text{NewP}_1; P_2 = \text{NewP}_2; \ldots \]
Endloop

Example:

\[ T(x,y) \leftarrow R(x,y) \]
\[ T(x,y) \leftarrow R(x,z), T(z,y) \]

\[ T(x,y) \leftarrow R(x,y) \cup \Pi_{xy}(R(x,z) \bowtie T(z,y)) \]

\[ T = \emptyset \]
Loop

\[ \text{NewT}(x,y) = R(x,y) \cup \Pi_{xy}(R(x,z) \bowtie T(z,y)) \]
if (New\(T = T\))
then exit
\[ T = \text{NewT} \]
Endloop
Discussion

• A datalog program *always* terminates (why?)

• What is the running time of a datalog program as a function of the input database?
Problem with the Naïve Algorithm

• The same facts are discovered over and over again

• The *semi-naïve* algorithm tries to reduce the number of facts discovered multiple times
Background: Incremental View Maintenance

Let $V$ be a view computed by one datalog rule (no recursion)

$$V ::= \text{body}$$

If (some of) the relations are updated: $R_1 \leftarrow R_1 \cup \Delta R_1$, $R_1 \leftarrow R_2 \cup \Delta R_2$, ...

Then the view is also modified as follows: $V \leftarrow V \cup \Delta V$

**Incremental view maintenance:**
Compute $\Delta V$ without having to recompute $V$
Background: Incremental View Maintenance

Example 1:

\[ V(x,y) :- R(x,z), S(z,y) \]

If \( R \leftarrow R \cup \Delta R \) then what is \( \Delta V(x,y) \)?
V(x,y) :- R(x,z),S(z,y)

If \( R \leftarrow R \cup \Delta R \) then what is \( \Delta V(x,y) \)?

\[ \Delta V(x,y) :- \Delta R(x,z), S(z,y) \]
Background: Incremental View Maintenance

Example 2:

\[ V(x,y) :\text{-} R(x,z),S(z,y) \]

If \( R \leftarrow R \cup \Delta R \) and \( S \leftarrow S \cup \Delta S \) then what is \( \Delta V(x,y) \) ?
Background: Incremental View Maintenance

Example 2:

\[
V(x,y) \leftarrow R(x,z), S(z,y)
\]

If \( R \leftarrow R \cup \Delta R \) and \( S \leftarrow S \cup \Delta S \) then what is \( \Delta V(x,y) \) ?

\[
\begin{align*}
\Delta V(x,y) & \leftarrow \Delta R(x,z), S(z,y) \\
\Delta V(x,y) & \leftarrow R(x,z), \Delta S(z,y) \\
\Delta V(x,y) & \leftarrow \Delta R(x,z), \Delta S(z,y)
\end{align*}
\]
Background: Incremental View Maintenance

Example 3:

\[ V(x,y) :- T(x,z), T(z,y) \]

If \( T \leftarrow T \cup \Delta T \), then what is \( \Delta V(x,y) \)?
Background: Incremental View
Maintenace

Example 3:

\[ V(x,y) :- T(x,z), T(z,y) \]

If \( T \leftarrow T \cup \Delta T \)
then what is \( \Delta V(x,y) \)?

\[ \Delta V(x,y) :- \Delta T(x,z), T(z,y) \]
\[ \Delta V(x,y) :- T(x,z), \Delta T(z,y) \]
\[ \Delta V(x,y) :- \Delta T(x,z), \Delta T(z,y) \]
Semi-naïve Evaluation Algorithm

Separate the Datalog program into the non-recursive, and the recursive part. Each $P_i$ defined by non-recursive-SPJU$_i$ and (recursive-)SPJU$_i$.

$$P_1 = \Delta P_1 = \text{non-recursive-SPJU}_1, P_2 = \Delta P_2 = \text{non-recursive-SPJU}_2, \ldots$$

**Loop**

$$\Delta P_1 = \Delta \text{SPJU}_1; \Delta P_2 = \Delta \text{SPJU}_2; \ldots$$

**if** $(\Delta P_1 = \emptyset \text{ and } \Delta P_2 = \emptyset \text{ and } \ldots)$

then break

$$P_1 = P_1 \cup \Delta P_1; P_2 = P_2 \cup \Delta P_2; \ldots$$

**Endloop**
Semi-naïve Evaluation Algorithm

Separate the Datalog program into the non-recursive, and the recursive part. Each $P_i$ defined by non-recursive-$SPJU_i$ and (recursive-)SPJU$_i$.

$$P_1 = \Delta P_1 = \text{non-recursive-}SPJU_1, \ P_2 = \Delta P_2 = \text{non-recursive-}SPJU_2, \ldots$$

Loop

$\Delta P_1 = \Delta SPJU_1; \ \Delta P_2 = \Delta SPJU_2; \ \ldots$

if (\(\Delta P_1 = \emptyset\) and \(\Delta P_2 = \emptyset\) and \(\ldots\))

then break

$P_1 = P_1 \cup \Delta P_1; \ P_2 = P_2 \cup \Delta P_2; \ \ldots$

Endloop

Example:

$$T(x,y) :- R(x,y)$$  
$$T(x,y) :- R(x,z), T(z,y)$$

$$T = \Delta T = ? \ (\text{non-recursive rule})$$

Loop

$\Delta T(x,y) = ? \ (\text{recursive } \Delta\text{-rule})$

if (\(\Delta T = \emptyset\))

then break

$T = T \cup \Delta T$

Endloop
Semi-naïve Evaluation Algorithm

Separate the Datalog program into the non-recursive, and the recursive part. Each $P_i$ defined by non-recursive-SPJU$_i$ and (recursive-)SPJU$_i$.

Example:

\[
T(x,y) \leftarrow R(x,y) \\
\text{T}(x,y) \leftarrow R(x,z), T(z,y)
\]

\[
T(x,y) = R(x,y), \quad \Delta T(x,y) = R(x,y) \\
\text{Loop} \\
\quad \Delta T(x,y) = R(x,z), \Delta T(z,y) \\
\quad \text{if} (\Delta T = \emptyset) \\
\quad \quad \text{then break} \\
\quad \text{T} = \text{T} \cup \Delta T \\
\text{Endloop}
\]
Semi-naïve Evaluation Algorithm

Separate the Datalog program into the non-recursive, and the recursive part. Each $P_i$ defined by non-recursive-$SPJU_i$ and (recursive-)SPJU$_i$.

$$P_1 = \Delta P_1 = \text{non-recursive-}SPJU_1, \ P_2 = \Delta P_2 = \text{non-recursive-}SPJU_2, \ ...$$

Loop

$$\Delta P_1 = \Delta SPJU_1; \ \Delta P_2 = \Delta SPJU_2; \ ...$$

if $$(\Delta P_1 = \emptyset \ \text{and} \ \Delta P_2 = \emptyset \ \text{and} \ ...)$$

then break

$$P_1 = P_1 \cup \Delta P_1; \ P_2 = P_2 \cup \Delta P_2; \ ...$$

Endloop

Example:

\[
\begin{align*}
T(x,y) & : - R(x,y) \\
T(x,y) & : - R(x,z), \ T(z,y)
\end{align*}
\]

Note: for any linear datalog programs, the semi-naïve algorithm has only one $\Delta$-rule for each rule!
Simple datalog programs

R encodes a graph

\[
\begin{align*}
T(x,y) & : \neg R(x,y) \\
T(x,y) & : \neg R(x,z), T(z,y)
\end{align*}
\]

Initially:

\[
T = R, \quad \Delta T = R
\]

Loop

\[
\Delta T(x,y) = R(x,z), \quad \Delta T(z,y)
\]

if (\(\Delta T = \emptyset\))

\[\text{then break}\]

\[T = T \cup \Delta T\]

Endloop
**Simple datalog programs**

R encodes a graph

\[
R = \begin{array}{ccc}
1 & 2 & 4 \\
1 & 4 & \\
2 & 1 & 3 \\
2 & 3 \\
3 & 4 \\
4 & 5 \\
\end{array}
\]

Initially:

\[
T = R, \quad \Delta T = R
\]

Loop

\[
\Delta T(x, y) = R(x, z), \quad \Delta T(z, y)
\]

if \( \Delta T = \emptyset \)
  then break

\[
T = T \cup \Delta T
\]

Endloop

First iteration:

\[
T = \begin{array}{ccc}
1 & 2 & 4 \\
1 & 4 & \\
2 & 1 & 3 \\
2 & 3 \\
3 & 4 \\
4 & 5 \\
\end{array}
\]

Paths of length 2:

\[
\Delta T = \begin{array}{ccc}
1 & 2 & 4 \\
1 & 4 & \\
2 & 1 & 3 \\
2 & 3 \\
3 & 4 \\
4 & 5 \\
\end{array}
\]

T =

\[
\begin{array}{ccc}
1 & 2 & 4 \\
1 & 4 & \\
2 & 1 & 3 \\
2 & 3 \\
3 & 4 \\
4 & 5 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 3 \\
1 & 5 \\
2 & 2 \\
2 & 4 \\
3 & 5 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 1 \\
1 & 3 \\
1 & 5 \\
2 & 2 \\
2 & 4 \\
3 & 5 \\
\end{array}
\]

\[
\begin{array}{ccc}
1 & 1 \\
1 & 3 \\
1 & 5 \\
2 & 2 \\
2 & 4 \\
3 & 5 \\
\end{array}
\]
**Simple datalog programs**

R encodes a graph

\[
\begin{align*}
T(x, y) & \leftarrow R(x, y) \\
T(x, y) & \leftarrow R(x, z), T(z, y)
\end{align*}
\]

Initially:

First iteration:

Second iteration:

\[
\begin{align*}
T = R, \quad \Delta T = R \\
\text{Loop} \\
\Delta T(x, y) = R(x, z), \Delta T(z, y) \\
\text{if } (\Delta T = \emptyset) \text{ then break} \\
T = T \cup \Delta T \\
\text{Endloop}
\end{align*}
\]
Simple datalog programs

R encodes a graph

\[ T(x,y) :\neg R(x,y) \]
\[ T(x,y) :\neg R(x,z), T(z,y) \]

R =

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

\[ \Delta T = \text{paths of length } 2 \]

\[ T = \]

\[ \Delta T = \text{paths of length } 3 \]

\[ T = \]

\[ \Delta T = \text{paths of length } 4 \]

Initially:

1. First iteration: \[ T = \]
2. Second iteration: \[ T = \]
3. Third iteration: \[ T = \]

**Initial**

1. \[ T = R, \Delta T = R \]
2. **Loop**
3. \[ \Delta T(x,y) = R(x,z), \Delta T(z,y) \]
4. **if** (\( \Delta T = \emptyset \))
5. **then** break
6. **Endloop**

\[ T = T \cup \Delta T \]
Discussion of Semi-Naïve Algorithm

• Avoids re-computing some tuples, but not all tuples
• Easy to implement, no disadvantage over naïve
• A rule is called *linear* if its body contains only one recursive IDB predicate:
  – A linear rule always results in a single incremental rule
  – A non-linear rule may result in multiple incremental rules
Summary of Datalog

• Simple syntax for expressing queries with recursion
• Bottom-up evaluation – always terminates
  – Naïve evaluation
  – Semi-naïve evaluation
• Next lecture:
  – Discuss the paper on datalog
  – Datalog semantics
  – Datalog with negation