CSE 544
Principles of Database Management Systems

Lectures 6: Datalog (2)

Reminders

- This Friday: project proposals due (turnin using git)
- Monday: paper review due (12h before lecture)
- Next Friday: brief meetings to discuss your project
- Next Friday: hw2 due

Suggested Readings for Datalog

- Joe Hellerstein, “The Declarative Imperative,” SIGMOD Record 2010
- R&G Chapter 24
- Phokion Kolaitis’ tutorial on database theory at Simon’s https://simons.berkeley.edu/sites/default/files/docs/5241/simons16-21.pdf
- Daniel Zinn, Todd J. Green, Bertram Ludäscher: Win-move is coordination-free (sometimes). ICDT 2012

Review

- What is datalog?
- What is the naïve evaluation algorithm?
- What is the seminaive algorithm?

Outline

- Semi-joins
- Semi-join reduction
- Acyclic queries
- Magic sets

Cost of Computing a Query

- Suppose |R| = |S| = n
- What is the cost of a join R ⋈ S?

$$q(x,y,z) = R(x,y), S(y,z)$$

- Algorithms (discuss in class):

Cost of Computing a Query

- Suppose |R| = |S| = n

- What is the cost of a join $R \bowtie S$?

$$q(x,y,z) = R(x,y), S(y,z)$$

- Algorithms (discuss in class):
  - Nested loop join
  - Hash join
  - Merge join

Cost of Computing a Query

- Suppose |R| = |S| = n

- What is the cost of a join R ⋈ S?

$$q(x,y,z) = R(x,y), S(y,z)$$

- Algorithms (discuss in class):
  - Nested loop join $O(n^2)$
  - Hash join $O(n) \dots O(n^2)$
  - Merge join $O(n \log n) \dots O(n^2)$

Key / foreign-key join

General case

Cost of Computing a Query

- Suppose |R| = |S| = |T| = |K| = n

- What is the complexity of computing these queries?

  Q1(x,y,z) = R(x,y), S(y,z) $O(n^2)$

Cost of Computing a Query

- Suppose |R| = |S| = |T| = |K| = n
- What is the complexity of computing these queries?

  Q1(x,y,z) = R(x,y), S(y,z) $O(n^2)$

  Q2(x,y,z,u) = R(x,y),S(y,z),T(z,u)

Cost of Computing a Query

- Suppose |R| = |S| = |T| = |K| = n

- What is the complexity of computing these queries?

  Q1(x,y,z) = R(x,y), S(y,z) $O(n^2)$

  Q2(x,y,z,u) = R(x,y),S(y,z),T(z,u) $O(n^3)$

Cost of Computing a Query

- Suppose |R| = |S| = |T| = |K| = n

- What is the complexity of computing these queries?

Q1(x,y,z) = R(x,y), S(y,z) $O(n^2)$

Q2(x,y,z,u) = R(x,y),S(y,z),T(z,u) $O(n^3)$

Q3(x,y,z,u,v) = R(x,y),S(y,z),T(z,u),K(u,v)

Cost of Computing a Query

- Suppose |R| = |S| = |T| = |K| = n

- What is the complexity of computing these queries?

Q1(x,y,z) = R(x,y), S(y,z) $O(n^2)$

Q2(x,y,z,u) = R(x,y),S(y,z),T(z,u) $O(n^3)$

Q3(x,y,z,u,v) = R(x,y),S(y,z),T(z,u),K(u,v) $O(n^4)$

Cost of Computing a Query

- Suppose |R| = |S| = |T| = |K| = n

- What is the complexity of computing these queries?

Q1(x,y,z) = R(x,y), S(y,z) $O(n^2)$

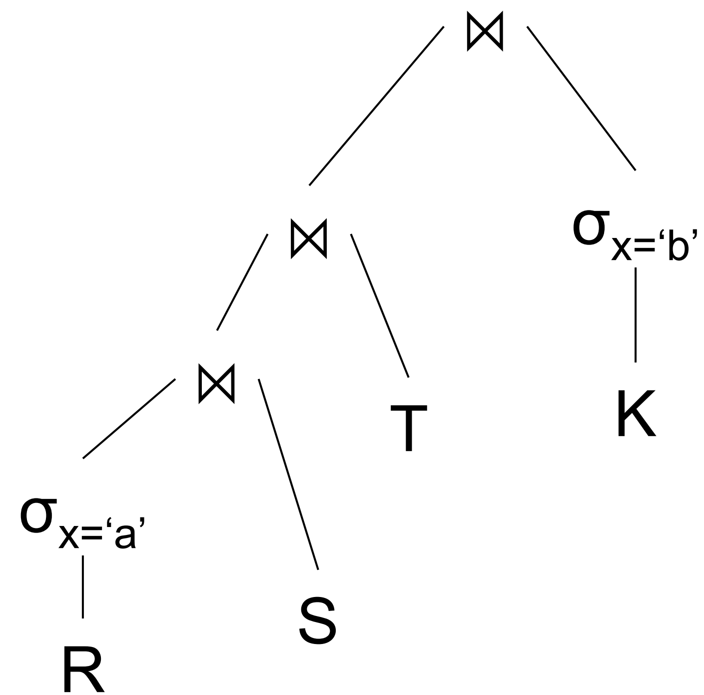
Q2(x,y,z,u) = R(x,y),S(y,z),T(z,u) $O(n^3)$

Q3(x,y,z,u,v) = R(x,y),S(y,z),T(z,u),K(u,v) $O(n^4)$

Ideally cost: O(|Input| + |Output|)

Cost of Computing a Query

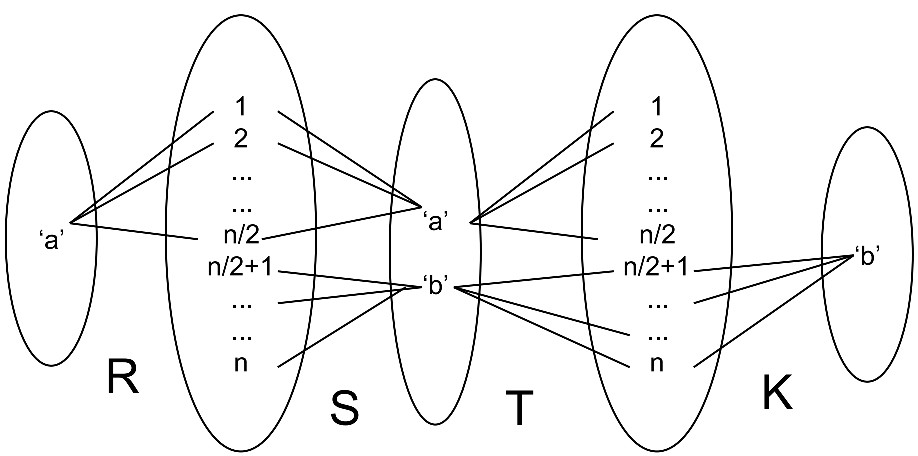
- Naïve computation often exceeds this bound
- Q(x,y,z,u) = R(‘a’, y), S(y,z), T(z,u), K(u,’b’)

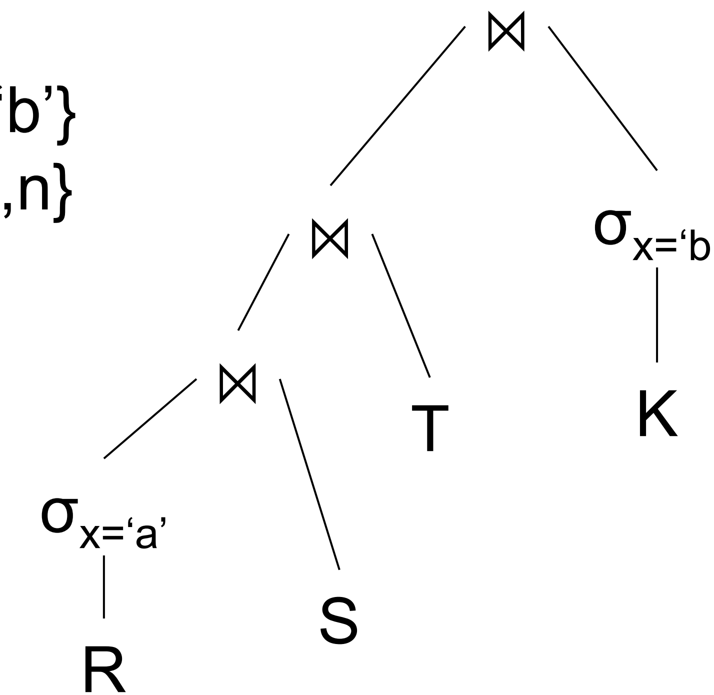


Cost of Computing a Query

- Naïve computation often exceeds this bound
- Q(x,y,z,u) = R('a', y), S(y,z), T(z,u), K(u,'b')

  R = {'a'}×{1,...,n/2}
  S = {1,...,n/2}×{'a'}∪{n/2+1,...,n}×{'b'}
  T = {'a'}×{1,...,n/2}∪{'b'}×{n/2+1,...,n}
  K = {n/2+1,...,n}×{'b'}

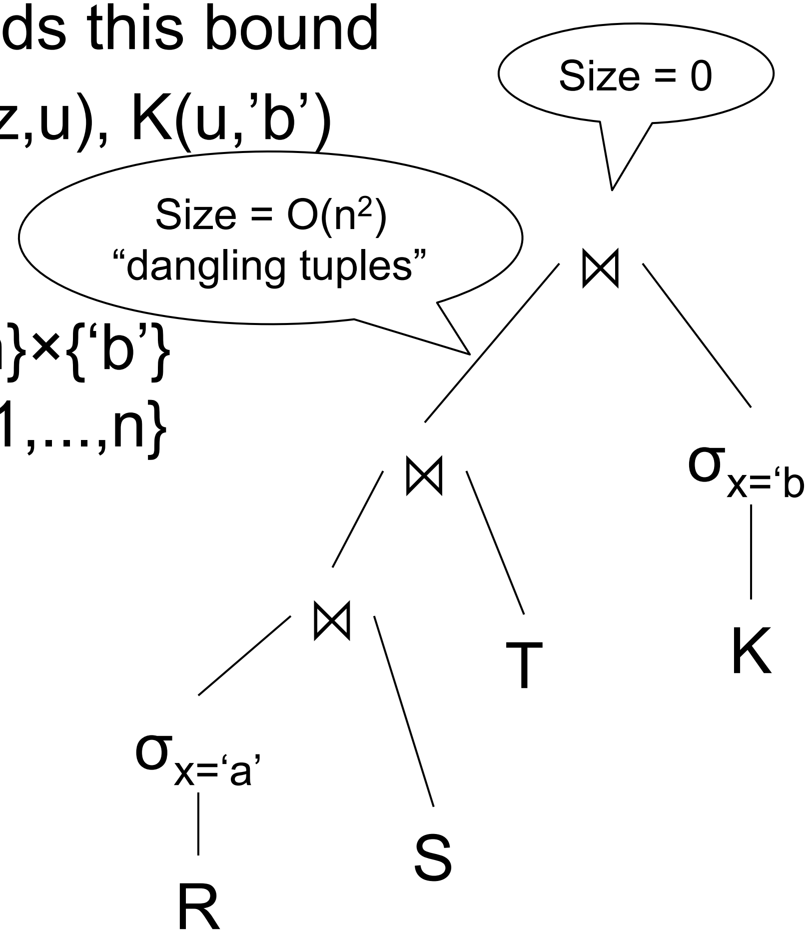


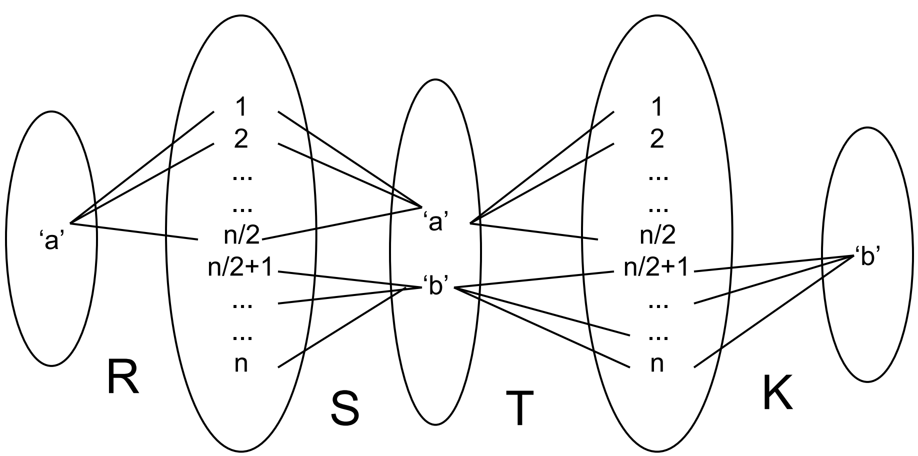


Cost of Computing a Query

- Naïve computation often exceeds this bound
- Q(x,y,z,u) = R('a', y), S(y,z), T(z,u), K(u,'b')

R = {'a'}×{1,...,n/2}
S = {1,...,n/2}×{'a'} ∪ {n/2+1,...,n}×{'b'}
T = {'a'}×{1,...,n/2} ∪ {'b'}×{n/2+1,...,n}
K = {n/2+1,...,n}×{'b'}





Cost ≠ O(|Input| + |Output|)

The Semijoin Operator

Definition: the semi-join operation is

$$R \ltimes S = \Pi_{Attr(R)}(R \bowtie S)$$

Properties of Semijoins

- $R(A,B) \ltimes S(B,C)$ same as $Q(A,B)$:- $R(A,B),S(B,C)$

- Cost: $O(|R| + |S|)$ (ignoring log-factors)

- Cost is independent on the join output

- The law of semijoins is:

$$R \bowtie S = (R \ltimes S) \bowtie S$$

Consequence: we can perform a semi-join before a join

Outline

- Semi-joins
- Semi-join reduction
- Acyclic queries
- Magic sets

Semijoin Optimizations

- In parallel databases: often combined with Bloom Filters (pp. 747 in the textbook)
- Magic sets for datalog were invented after semi-join reductions, and the connection became clear only later
- Some complex semi-join reductions for non-recursive SQL optimizations are sometimes called “magic sets”

Semijoin Reducer

- Given a query:

$$Q = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$$

- A *semijoin reducer* for Q is

$$R_{i1} = R_{i1} \ltimes R_{j1}$$
$$R_{i2} = R_{i2} \ltimes R_{j2}$$
$$\ldots\ldots$$
$$R_{ip} = R_{ip} \ltimes R_{jp}$$

such that the query is equivalent to:

$$Q = R_{k1} \bowtie R_{k2} \bowtie \ldots \bowtie R_{kn}$$

- A *full reducer* is such that no dangling tuples remain

Example

- Example:

$$Q = R(A,B) \bowtie S(B,C)$$

- A semijoin reducer is:

$$R_1(A,B) = R(A,B) \ltimes S(B,C)$$

- The rewritten query is:

$$Q = R_1(A,B) \bowtie S(B,C)$$

Semijoin Reducer

- More complex example:

  Q(y,z,u) = R('a', y), S(y,z), T(z,u), K(u,'b')

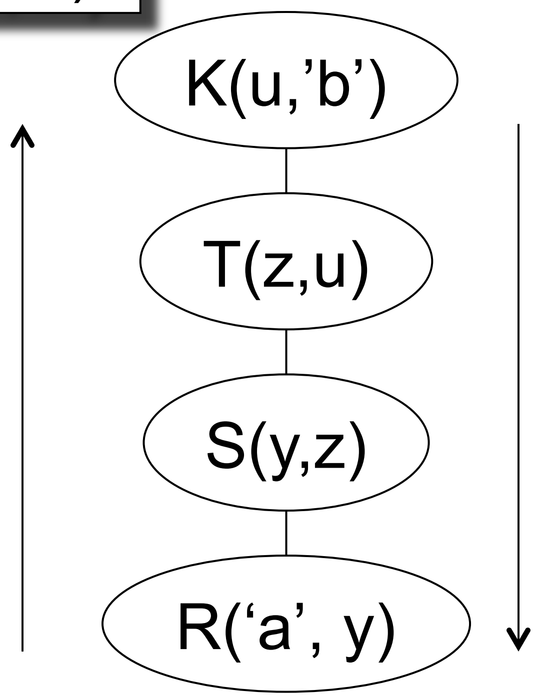
- Find a full reducer

Semijoin Reducer

- More complex example:

  Q(y,z,u) = R('a', y), S(y,z), T(z,u), K(u,'b')

- Find a full reducer



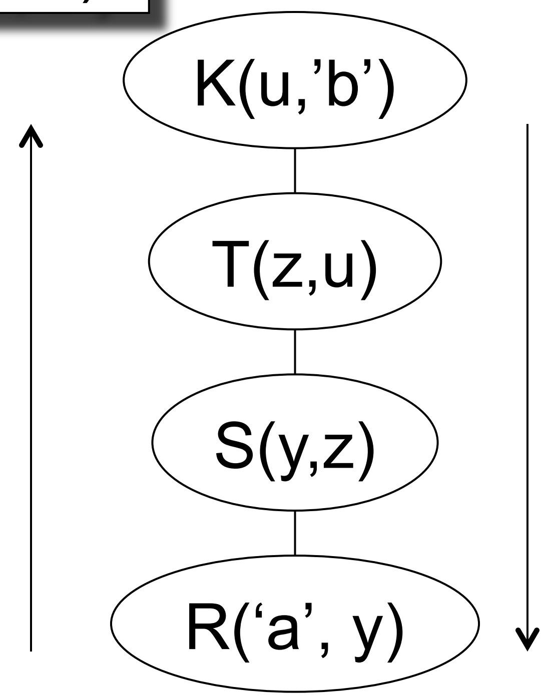
Semijoin Reducer

- More complex example:

Q(y,z,u) = R(‘a’, y), S(y,z), T(z,u), K(u,’b’)

- Find a full reducer

S’(y,z) :- S(y,z) $\ltimes$ R(‘a’, y)
T’(z,u) :- T(z,u) $\ltimes$ S’(y,z)
K’(u) :- K(u,’b’) $\ltimes$ T’(z,u)
T’’(z,u) :- T’(z,u) $\ltimes$ K’(u)
S’’(y,z) :- S’(y,z) $\ltimes$ T’’(z,u)
R’’(y) :- R(‘a’,y) $\ltimes$ S’’(y,z)



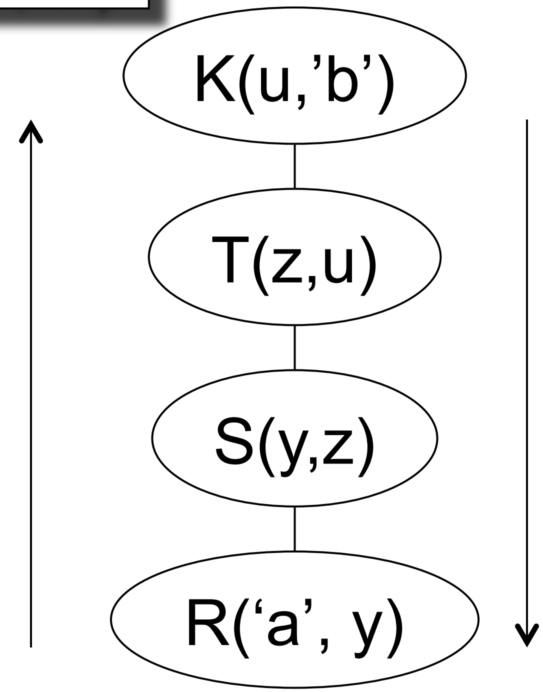
Semijoin Reducer

- More complex example:

Q(y,z,u) = R(‘a’, y), S(y,z), T(z,u), K(u,’b’)

- Find a full reducer

S’(y,z) :- S(y,z) $\ltimes$ R(‘a’, y)
T’(z,u) :- T(z,u) $\ltimes$ S’(y,z)
K’(u) :- K(u,’b’) $\ltimes$ T’(z,u)
T’’(z,u) :- T’(z,u) $\ltimes$ K’(u)
S’’(y,z) :- S’(y,z) $\ltimes$ T’’(z,u)
R’’(y) :- R(‘a’,y) $\ltimes$ S’’(y,z)



- Finally, compute:

Q(y,z,u) = R’’(y), S’’(y,z), T’’(z,u), K’’(u)

Practice at Home...

- Find semi-join reducer for R(x,y),S(y,z),T(z,u),K(u,v),L(v,w)

Not All Queries Have Full Reducers

- Example:

$$Q = R(A,B) \bowtie S(B,C) \bowtie T(A,C)$$

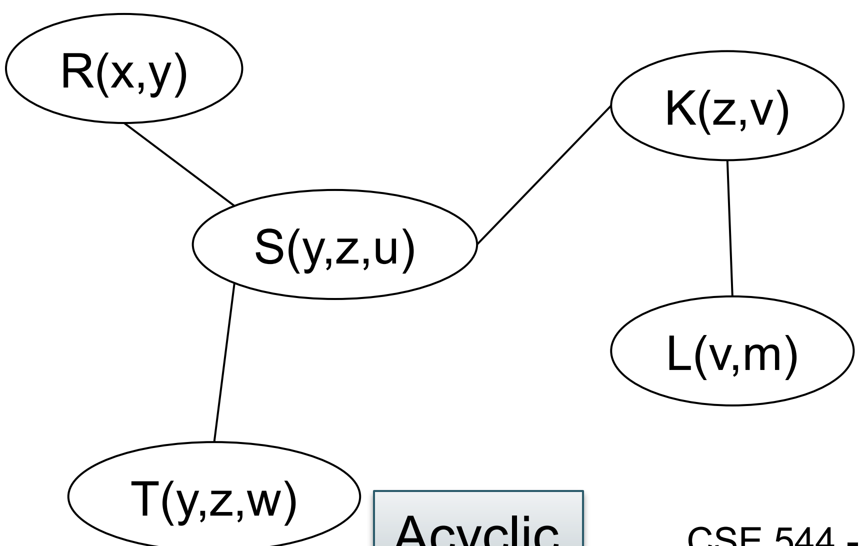
- Can write many different semi-join reducers
- But no full reducer of length O(1) exists

Outline

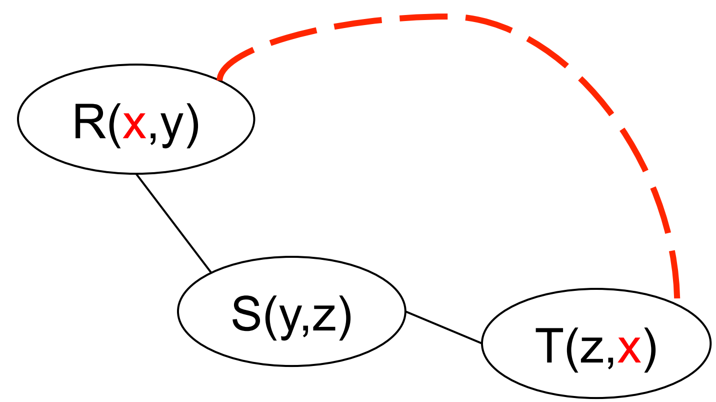
- Semi-joins
- Semi-join reduction
- Acyclic queries
- Magic sets

Acyclic Queries

- Fix a Conjunctive Query without self-joins
- Q is *acyclic* if its atoms can be organized in a tree such that for every variable the set of nodes that contain that variable form a connected component



Acyclic



R(x,y),S(y,z),T(z,x) is cyclic

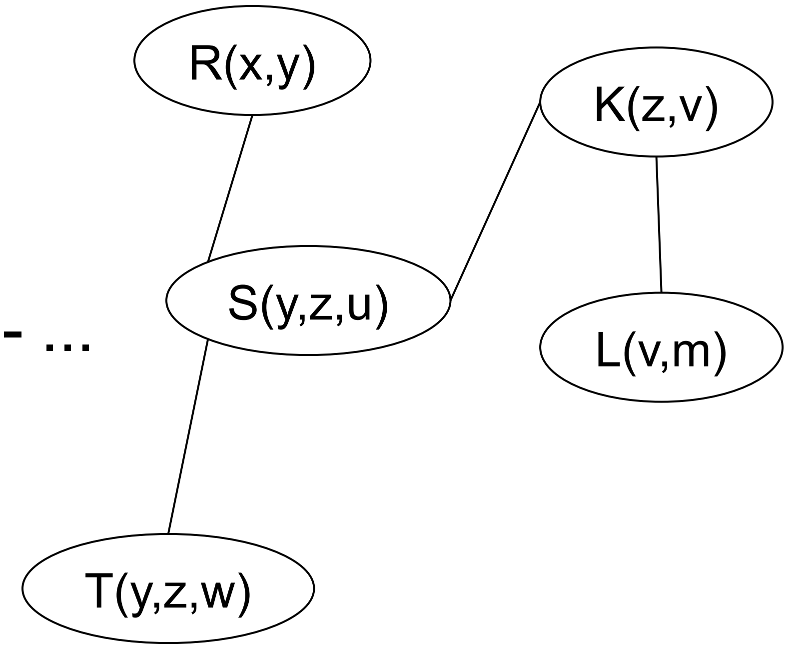
Yannakakis Algorithm

- Given: acyclic query Q
- Compute Q on any database in time O(|Input|+|Output|)

- Step 1: semi-join reduction
  – Pick any root node x in the tree decomposition of Q
  – Do a semi-join reduction sweep from the leaves to x
  – Do a semi-join reduction sweep from x to the leaves
- Step 2: compute the joins bottom up, with early projections

Examples in Class

- Boolean query: Q() :- ...
- Non-boolean: Q(x,m) :- ...
- With aggregate: Q(x,sum(m)) :- ...
- And also: Q(x,count(*)) :- ...



In all cases: runtime = O(|R|+|S|+...+|L| + |Output|)

Testing if Q is Acyclic

- An *ear* of Q is an atom R(X) with the following property:
  - Let X' ⊆ X be the set of join variables (meaning: they occur in at least one other atom)
  - There exists some other atom S(Y) such that X' ⊆ Y
- The GYO algorithm (Graham,Yu,Özsoyoğlu) for testing if Q is acyclic:
  - While Q has an ear R(X), remove the atom R(X) from the query
  - If all atoms were removed, then Q is acyclic
  - If atoms remain but there is no ear, then Q is cyclic
- Show example in class

Outline

- Semi-joins
- Semi-join reduction
- Acyclic queries
- Magic sets

Magic Sets

- Problem: datalog programs compute *a lot*,
  but sometimes we need only *very little*

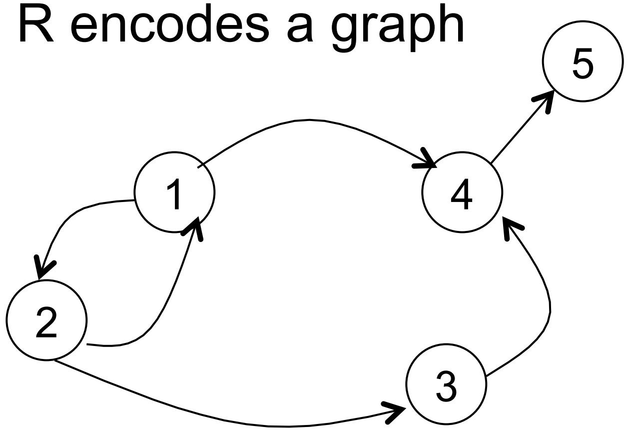
- Prolog computes top-down and retrieves *very little*
  datalog computes bottom up retrieves *a lot*

- (Prolog has other issues: left recursive prolog never terminates!)

- Magic sets transform a datalog program P into a new program P', such that bottom-up(P') = top-down(P)

Example 1

R encodes a graph



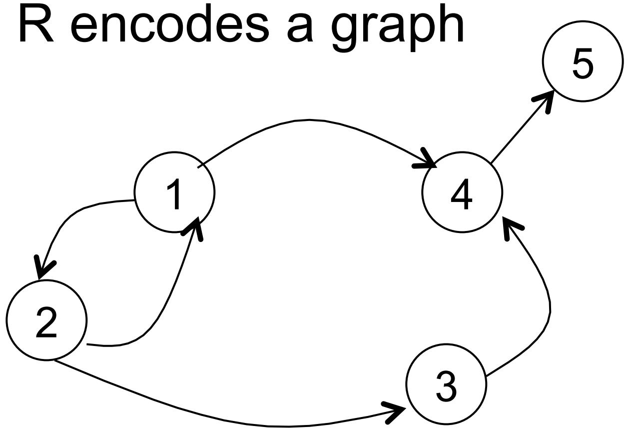

```
T(x,y) :- E(x,y)
T(x,y) :- T(x,z),E(z,y)
Q(y)   :- T(3,y)
```

a constant

Bottom-up evaluation very inefficient

Example 1

R encodes a graph




```
T(x,y) :- E(x,y)
T(x,y) :- T(x,z),E(z,y)
Q(y)   :- T(3,y)
```

a constant

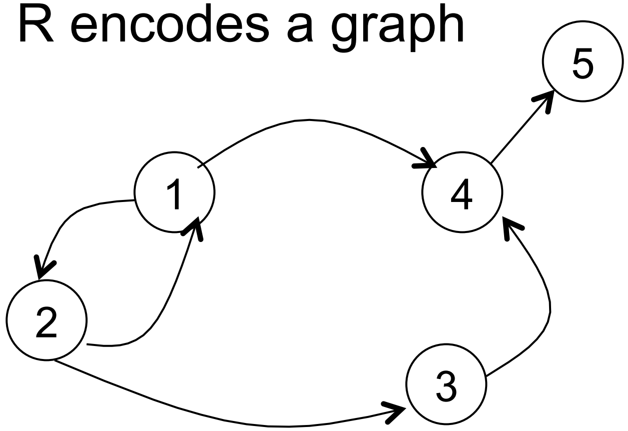
Manual optimization:

```
Q(y) :- E(3,y)
Q(y) :- Q(x),E(x,y)
```

Bottom-up evaluation very inefficient

Example 2

R encodes a graph



Same generation

SG(x,x) :- V(x)
SG(x,y) :- Up(x,u),SG(u,v),Dn(u,y)
Q(y) :- SG(1,y)

If we define
Up(a,b) = E(b,a)
Dn(a,b) = E(a,b)
then SG = “same generation”

Manual optimization???

Magic Set Rewriting (simplified)

- For each IDB predicate create “adorned” versions, with binding patters
- For each adorned IDB P, create a predicate $Magic_P$
- For each rule, create several rules, one for each possible adornment of the head:
  - Allow information to flow left-to-right (“sideways information passing”), and this defines the required adornments of the IDB’s in the body
  - If there are k IDB’s in the body, create k+1 supplementary relations $Supp_i$, which guard the set of bound variables passed on to the i’th IDB
- New rules defining $Magic_P$: one for the query, and one for each $Supp_i$ preceding an occurrence of P in a body

Adorned predicate

- b=bound, f=free
- $T^{bf}(x,y)$ means:
  - The values of x are known
  - The values of y are not known (need to be retrieved)
- Need to create all combinations: T^{bf}, T^{fb}

- Side-ways information passing means that we adorn rules allowing information to flow left-to-right

  - E.g. $T(x,y)$:- $E(x,u),T(u,v),E(v,w),T(w,z),E(z,y)$

  - Adorned: $T^{bf}(x,y)$:- $E(x,u),T^{bf}(u,v),E(v,w),T^{bf}(w,z),E(z,y)$

Supplementary Relations

- Given adornment $T^{bf}(x,y)$, a new predicate Supp(x) contains the (small!) set of values x for which we want to compute $T^{bf}(x,y)$
- E.g. $T^{bf}(x,y)$:- $E(x,u), T^{bf}(u,v), E(v,w), T^{bf}(w,z), E(z,y)$

$Supp_0(x)$

$Supp_1(x,u)$

$Supp_2(x,w)$

$Supp_3(x,y)$

Supp Rules

- E.g. $T^{bf}(x,y)$:- $E(x,u),T^{bf}(u,v),E(v,w),T^{bf}(w,z),E(z,y)$

$Supp_0(x)$

$Supp_1(x,u)$

$Supp_2(x,w)$

$Supp_3(x,y)$

Becomes:

- $Supp_0(x)$:- $Magic_{Tbf}(x)$ /* next slide … */
- $Supp_1(x,u)$:- $Supp_0(x)$, $E(x,u)$
- $Supp_2(x,w)$:- $Supp_1(x,u)$, $T^{bf}(u,v),E(v,w)$
- $Supp_3(x,y)$:- $Supp_2(x,w)$, $T^{bf}(w,z),E(z,y)$
- $T^{bf}(x,y)$:- $Supp_3(x,y)$

$Supp_0$ and $Supp_3$ are redundant

Adding the Magic Predicate

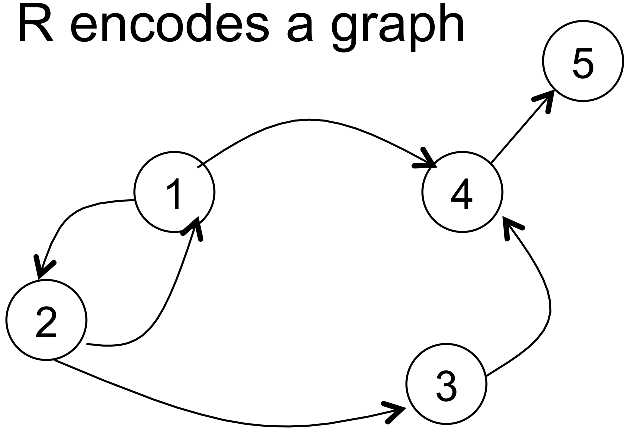
- E.g. $T^{bf}(x,y)$:- $E(x,u), T^{bf}(u,v), E(v,w), T^{bf}(w,z), E(z,y)$

  $Supp_0(x)$ $Supp_1(x,u)$ $Supp_2(x,w)$ $Supp_3(x,y)$

- $Magic_{Tbf}(x)$ = the set of bounded values of x for which we need to compute $T^{bf}(x,y)$
- E.g.
  - $Magic_{Tbf}(3)$:- /* if the query is Q(y) :- T(3,y) */
  - $Magic_{Tbf}(u)$:- $Supp_1(x,u)$ /* need to compute $T^{bf}(u,v)$ */
  - $Magic_{Tbf}(w)$:- $Supp_2(x,w)$ /* need to compute $T^{bf}(w,z)$ */

Example 1

R encodes a graph



Magic Sets

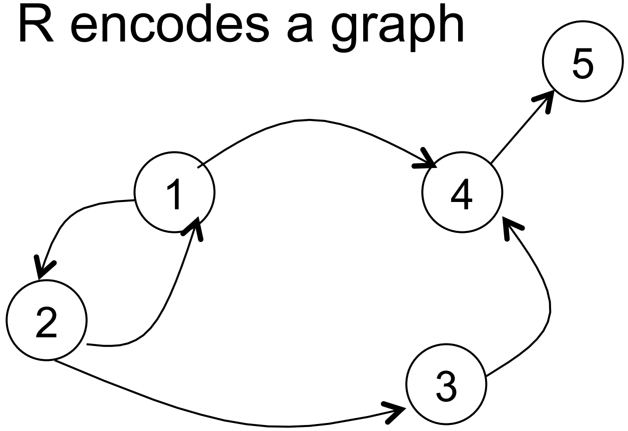
Original:

```
T(x,y) :- E(x,y)
T(x,y) :- T(x,z),E(z,y)
Q(y)   :- T(3,y)
```

Adorned:

Example 1

R encodes a graph



Magic Sets

Original:

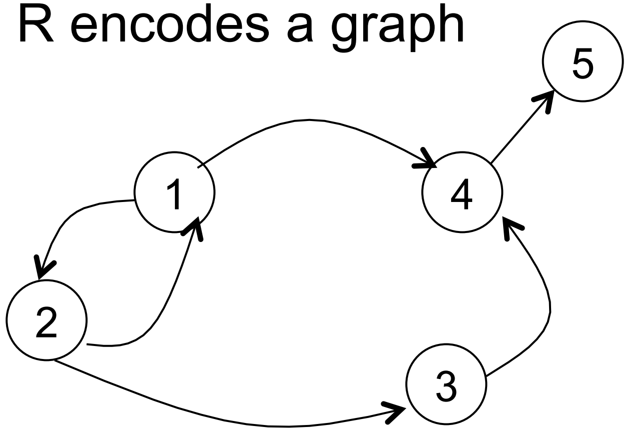
$T(x,y)$:- $E(x,y)$
$T(x,y)$:- $T(x,z),E(z,y)$
$Q(y)$:- $T(3,y)$

Adorned:

$T^{bf}(x,y)$:- $E(x,y)$
$T^{bf}(x,y)$:- $T^{bf}(x,z),E(z,y)$
$Q(y)$:- $T^{bf}(3,y)$

Example 1

R encodes a graph



Original:

$T(x,y)$:- $E(x,y)$
$T(x,y)$:- $T(x,z),E(z,y)$
$Q(y)$:- $T(3,y)$

Adorned:

$T^{bf}(x,y)$:- $E(x,y)$
$T^{bf}(x,y)$:- $T^{bf}(x,z),E(z,y)$
$Q(y)$:- $T^{bf}(3,y)$

Magic Sets

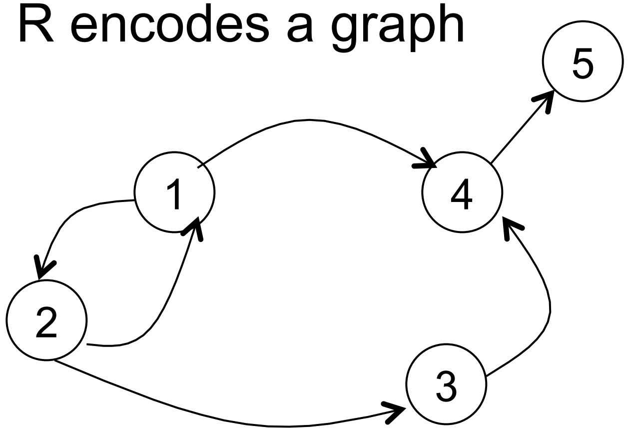
/* T(x,y) :- E(x,y) */
$Supp_0(x)$:- $Magic_{Tbf}(x)$
$Supp_1(x,y)$:- $Supp_0(x),E(x,y)$
$T^{bf}(x,y)$:- $Supp_1(x,y)$

/* T(x,y) :- T(x,z),E(z,y) */
$Supp'_0(x)$:- $Magic_{Tbf}(x)$
$Supp'_1(x,z)$:- $Supp'_0(x), T^{bf}(x,z)$
$Supp'_2(x,y)$:- $Supp'_1(x,z), E(z,y)$
$T^{bf}(x,y)$:- $Supp'_2(x,y)$

/* Q(y) :- T(3,y) */
$Magic_{Tbf}(3)$:-
$Magic_{Tbf}(x)$:- $Supp'_0(x)$ /* redundant */

Practice at home

R encodes a graph



We saw this

```
T(x,y) :- E(x,y)
T(x,y) :- T(x,z),E(z,y)
Q(y)   :- T(3,y)
```

```
T(x,y) :- E(x,y)
T(x,y) :- E(x,z),T(z,y)
Q(y)   :- T(3,y)
```

```
T(x,y) :- E(x,y)
T(x,y) :- T(x,z),T(z,y)
Q(y)   :- T(3,y)
```