Query Optimization: Exercise

Session 1

Maximilian Rieger

October 20, 2023

Algebra Revised

uni schema:

- Studenten : $\{[\underline{\text{MatrNr: integer}}, \text{Name: string, Semester: integer}]\}$
- Vorlesungen : $\{[\underline{\text{VorlNr: integer}}, \text{Titel: string, SWS: integer, gelesenVon: integer}]\}$
- Professoren : $\{[\underline{\text{PersNr: integer}}, \text{Name: string, Rang: string, Raum: integer}]\}$
- Assistenten : $\{[\underline{\text{PersNr: integer}}, \text{Name: string, Fachgebiet: string, Boss: integer}]\}$
- hoeren : $\{[\underline{\text{MatrNr: integer, VorlNr: integer}}]\}$
- voraussetzen : $\{[\underline{\text{Vorgaenger: integer, Nachfolger: integer}}]\}$
- pruefen : $\{[\underline{\text{MatrNr: integer, VorlNr: integer}}, \text{PersNr: integer, Note: decimal}]\}$

Relational Calculus

- *what* the result looks like (declarative)
- tuple calculus: $\{t|P(t)\}$
  - $\{p|p \in \text{Professoren} \wedge p.Rang = \text{'C4'}\}$
  - $\{s|s \in \text{Studenten}$

    $\wedge \exists h \in \text{hoeren}(s.MatrNr = h.MatrNr$

    $\wedge \exists v \in \text{Vorlesungen}(h.VorlNr = v.VorlNr$

    $\wedge \exists p \in \text{Professoren}(p.PersNr = v.gelesenVon \wedge p.Name = \text{'Curie'})))\}$
- domain calculus: $\{[v_1, ..., v_n]|P(v_1, ..., v_n)\}$
  - $\{[p, n, r, o]|[p, n, r, o] \in \text{Professoren} \wedge r = \text{'C4'}\}$
  - $\{[m, n, s]|[m, n, s] \in \text{Studenten}$

    $\wedge \exists v([m, v] \in \text{hoeren}$

    $\wedge \exists t, d, p([v, t, d, p] \in \text{Vorlesungen}$

    $\wedge \exists a, r, o([p, a, r, o] \in \text{Professoren} \wedge a = \text{'Curie'})))\}$

Relational Algebra

- *how* the result is built (procedural)
  - $\sigma_{Rang=\text{'C4'}}(\text{Professoren})$
  - $\sigma_{S.MatrNr=H.MatrNr}(S \times \sigma_{H.VorlNr=V.VorlNr}(H \times \sigma_{V.gelesenVon=P.PersNr}(V \times \sigma_{P.Name=\text{'Curie'}}(P))))$
  - $S \bowtie (H \bowtie (V \bowtie_{V.gelesenVon=P.PersNr} \sigma_{P.Name=\text{'Curie'}}(P)))$

Query Optimization: Exercise

Session 2

Altan Birler

October 30, 2023

Homework

Exercise 1

- Find all students that attended the lectures together with '*Schopenhauer*', excluding *Schopenhauer* himself.
  - SQL

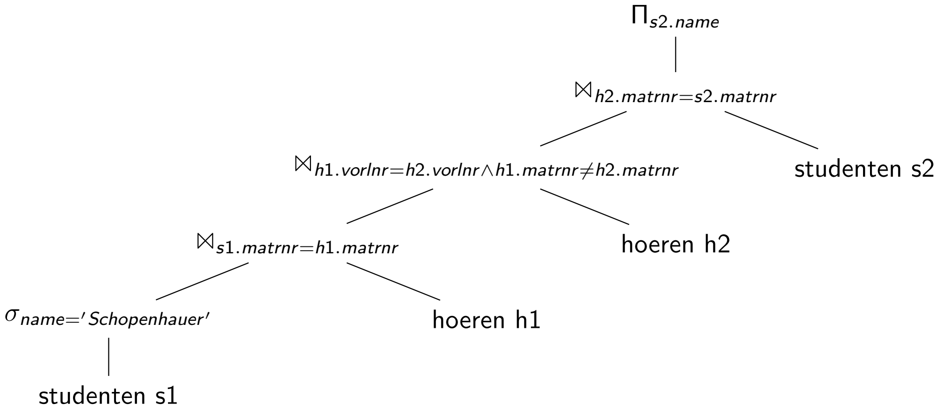
    ```
    select distinct s2.name
    from studenten s1, hoeren h1, hoeren h2, studenten s2
    where s1.name='Schopenhauer' and s1.matrnr=h1.matrnr
      and h1.vorlnr=h2.vorlnr and h2.matrnr=s2.matrnr
      and h1.matrnr<>h2.matrnr
    ```
  - tuple calculus

    $$\begin{aligned}\{s_1 | s_1 \in \text{Studenten} \wedge &\exists h_1 \in \text{hoeren}(s_1.MatrNr = h_1.MatrNr\\ \wedge &\exists h_2 \in \text{hoeren}(h_1.VorlNr = h_2.VorlNr \wedge h_1.MatrNr \neq h_2.MatrNr\\ \wedge &\exists s_2 \in \text{Studenten}(h_2.MatrNr = s_2.MatrNr \wedge h_2.Name = \text{'Schopenhauer'})\\ &))\}\end{aligned}$$
  - Why are they equivalent?

- Find all students that have ever attended a lecture together with *Schopenhauer*, excluding *Schopenhauer* himself.
  - domain calculus

$$\{[n_1] | \exists m_1, s_1([m_1, n_1, s_1] \in \text{Studenten}$$
$$\wedge \exists v([m_1, v] \in \text{hoeren}$$
$$\wedge \exists m_2([m_2, v] \in \text{hoeren} \wedge m_2 \neq m_1$$
$$\wedge \exists n_2, s_2([m_2, n_2, s_2] \in \text{Studenten} \wedge n_2 = \text{'Schopenhauer'})$$
$$)))\}$$

- Find all students that have ever attended a lecture together with '*Schopenhauer*', excluding *Schopenhauer* himself.
  - relational algebra


$\Pi_{s2.name}$
$\bowtie_{h2.matrnr=s2.matrnr}$
$\bowtie_{h1.vorlnr=h2.vorlnr \wedge h1.matrnr \neq h2.matrnr}$
studenten s2
$\bowtie_{s1.matrnr=h1.matrnr}$
hoeren h2
$\sigma_{name='Schopenhauer'}$
hoeren h1
studenten s1

- Find all professors whose lectures were all attended by at least two students.
  - SQL

```
select *
from professoren p
where not exists (
    select *
    from vorlesungen v
    where v.gelesenVon = p.persnr
        and not exists (
            select *
            from hoeren h1
            where h1.vorlnr = v.vorlnr
                and exists (
                    select * from hoeren h2
                    where h2.vorlnr = v.vorlnr and h2.matrnr <> h1.matrnr
                )
        )
)
```

Relational Algebra & SQL

algebra	name	SQL
$\sigma_p(L)$	selection	`select * from L where p`
$L \times R$	cross product	`L cross join R`
$L \bowtie_p R$	join	`L join R on p`
$L ⟕_p R$	left outer join	`L left join R on p`
$L ⟗_p R$	full outer join	`L full join R on p`
$L \ltimes_p R$	left semi join	`L where exists (select 1 from R where p)`
$L \triangleright_p R$	left anti join	`L where not exists (select 1 from R where p)`
$\Gamma_{k;r:agg(v)}(L)$	group by	`select k, agg(v) as r from L group by k`
$L - R$	set difference	L `except` R
$L -^+ R$	bag difference	L `except all` R
$L \cup R$	set union	L `union` R
$L \cup^+ R$	bag union	L `union all` R
$L \cap R$	set intersection	L `intersect` R
$L \cap^+ R$	bag intersection	L `intersect all` R

Multi-sets

- $\{\{a, b, b, c, c, c\}\} = \{a^1, b^2, c^3\} = \{(a, 1), (b, 2), (c, 3)\}$

Query Optimization: Exercise

Session 3

Maximilian Rieger

November 10, 2023

Homework

Exercise 1

- $\sigma_{p_1}(R_1 \bowtie_{p_2} R_2) \overset{?}{\equiv} \sigma_{p_1}(R_1) \bowtie_{p_2} R_2$
- $\sigma_{p_1}(R_1 \leftouterjoin_{p_2} R_2) \overset{?}{\equiv} \sigma_{p_1}(R_1) \leftouterjoin_{p_2} R_2$
- $\sigma_{p_1}(R_1 \rightouterjoin_{p_2} R_2) \overset{?}{\equiv} \sigma_{p_1}(R_1) \rightouterjoin_{p_2} R_2$
- $\sigma_{p_1}(R_1 \rightouterjoin_{p_2} R_2) \overset{?}{\equiv} R_1 \rightouterjoin_{p_1 \wedge p_2} R_2$
- $\sigma_{p_1}(R_1 \cup \rho_{x \to a, y \to b}(R_2)) \overset{?}{\equiv} \sigma_{p_1}(R_1) \cup \sigma_{p_1}(\rho_{x \to a, y \to b}(R_2))$
- $\sigma_{p_1}(R_1 \cap \rho_{x \to a, y \to b}(R_2)) \overset{?}{\equiv} \sigma_{p_1}(R_1) \cap \sigma_{p_1}(\rho_{x \to a, y \to b}(R_2))$

$$\sigma_{p_1}(R_1 \bowtie_{p_2} R_2) \overset{?}{\equiv} \sigma_{p_1}(R_1) \bowtie_{p_2} R_2 \text{ if } \mathcal{F}(p_1) \subseteq \mathcal{A}(R_1)$$

$$\begin{aligned}
\text{Let } t \in \sigma_{p_1}(R_1 \bowtie_{p_2} R_2) \quad &\Leftrightarrow \quad t \in (R_1 \bowtie_{p_2} R_2) \wedge p_1(t) \\
&\Leftrightarrow \quad \exists t_1 \in R_1, t_2 \in R_2 \text{ s.t. } t_1 \circ t_2 = t \wedge p_1(t) \wedge p_2(t) \\
&\overset{\mathcal{F}(p_1) \subseteq \mathcal{A}(R_1)}{\Leftrightarrow} \quad \exists t_1 \in R_1, t_2 \in R_2 \text{ s.t. } t_1 \circ t_2 = t \wedge p_1(t_1) \wedge p_2(t) \\
&\Leftrightarrow \quad \exists t_1 \in \sigma_{p_1}(R_1), t_2 \in R_2 \text{ s.t. } t_1 \circ t_2 = t \wedge p_2(t) \\
&\Leftrightarrow \quad t \in \sigma_{p_1}(R_1) \bowtie_{p_2} R_2
\end{aligned}$$

- $\sigma_{p_1}(R_1 \bowtie_{p_2} R_2) \overset{?}{\equiv} \sigma_{p_1}(R_1) \bowtie_{p_2} R_2$
- $\sigma_{p_1}(R_1 \leftouterjoin_{p_2} R_2) \overset{?}{\equiv} \sigma_{p_1}(R_1) \leftouterjoin_{p_2} R_2$
- $\sigma_{p_1}(R_1 \rightouterjoin_{p_2} R_2) \overset{?}{\equiv} \sigma_{p_1}(R_1) \rightouterjoin_{p_2} R_2$
- $\sigma_{p_1}(R_1 \rightouterjoin_{p_2} R_2) \overset{?}{\equiv} R_1 \rightouterjoin_{p_1 \wedge p_2} R_2$
- $\sigma_{p_1}(R_1 \cup \rho_{x \rightarrow a, y \rightarrow b}(R_2)) \overset{?}{\equiv} \sigma_{p_1}(R_1) \cup \sigma_{p_1}(\rho_{x \rightarrow a, y \rightarrow b}(R_2))$
- $\sigma_{p_1}(R_1 \cap \rho_{x \rightarrow a, y \rightarrow b}(R_2)) \overset{?}{\equiv} \sigma_{p_1}(R_1) \cap \sigma_{p_1}(\rho_{x \rightarrow a, y \rightarrow b}(R_2))$

Exercise 2

- $|R| = 1,000$ pages, $|S| = 100,000$ pages
- 1 page $l_p = 50$ tuples, 1 block $l_b = 100$ pages
- avg. access $t_s = 10$ ms, transfer speed $r = 10,000$ pages/sec
- Time for (blockwise) nested loops join?
- We assume we can read a whole relation with only one seek.

$$t_{NL} = |R| l_p \left(t_s + |S| \frac{1}{r} \right)$$

$$t_{BNL} = \frac{|R|}{l_b} \left(t_s + |S| \frac{1}{r} \right)$$

Exercise 3

$$\begin{aligned} L \cap R &= \{(t, min(L(t), R(t))) \mid t \in dom(L) \cap dom(R)\} \\ L \cup R &= \{(t, L(t) + R(t)) \mid t \in dom(L) \cup dom(R)\} \\ L - R &= \{(t, max(L(t) - R(t), 0)) \mid t \in dom(L)\} \end{aligned}$$

Show:

$$L \cap R = L - (L - R)$$

$$\begin{aligned} L - (L - R) &= L - \{(t, max(L(t) - R(t), 0)) \mid t \in dom(L)\} \\ &= \{(t, max(L(t) - max(L(t) - R(t), 0), 0)) \mid t \in dom(L)\} \end{aligned}$$

$$L - (L - R) = \{(t, max(L(t) - max(L(t) - R(t), 0), 0)) \mid t \in dom(L)\}$$

$$max(L(t) - max(L(t) - R(t), 0), 0)$$

For $L(t) \geq R(t)$:

$$max(L(t) - max(L(t) - R(t), 0), 0) = max(L(t) - L(t) + R(t), 0) = R(t)$$

For $L(t) < R(t)$:

$$max(L(t) - max(L(t) - R(t), 0), 0) = max(L(t) - 0, 0) = L(t)$$

$$max(L(t) - max(L(t) - R(t), 0), 0) = min(L(t), R(t))$$

$$\begin{aligned} L - (L - R) &= \{(t, min(L(t), R(t))) \mid t \in dom(L)\} \\ &= \{(t, min(L(t), R(t))) \mid t \in dom(L) \cap dom(R)\} \end{aligned}$$

$$t \in dom(L) \wedge t \notin dom(R) \Rightarrow min(L(t), R(t)) = 0$$

$$L ⟗_p R = ?$$

$$L \bowtie_p R = \{(t, L(t_1) \cdot R(t_2)) \mid t = t_1 \circ t_2 \land p(t) \land t_1 \in dom(L) \land t_2 \in dom(R)\}$$

For tuples in L but not R:

$$\{(t, L(t_1) \mid t = t_1 \circ t_2 \land p(t) \land t_1 \in dom(L) \land t_2 = \circ_{a \in \mathcal{A}(R)}(a : null)\}$$

$$\begin{aligned} L ⟗_p R = \{(t, max(L(t_1) \cdot R(t_2), L(t_1), R(t_2))) \mid t = t_1 \circ t_2 \land p(t) \land \\ t_1 \in dom(L) \cup \circ_{a \in \mathcal{A}(L)}(a : null) \land t_2 \in dom(R) \cup \circ_{a \in \mathcal{A}(R)}(a : null)\} \end{aligned}$$

Selectivity

- cardinalities $|R_i|$
- selectivities $f_{i,j}$: if $p_{i,j}$ is the join predicate between R_i and R_j, define

$$f_{i,j} = \frac{|R_i \bowtie_{p_{i,j}} R_j|}{|R_i \times R_j|}$$

- the more selective a join is, the lower the selectivity
- selectivity is estimated by dbms, used to estimate cardinalities of intermediate results

Textbook Optimization

- Translate SQL into an executable plan
- Find all Students that attend the course 'Ethik'
  - SQL query
  - canonical translation
  - break up conjunctive selections
  - push down selections
  - introduce joins
  - determine join order
  - introduce and push down projections

Query Optimization: Exercise

Session 4

Altan Birler

November 17, 2023

Exercise 1

All cardinality estimates make certain assumptions about data. We would like to make as few assumptions as possible, but sometimes we have no other options. Here are some common assumptions from query optimization folklore that we will utilize

- Independence: Data distributions of different columns are independent.
- Uniformity: Values in a column are distributed uniformly.
- Containment: Smaller set is contained within the larger set. “Users query for data that exists.” [1]

We prefer to make as few assumptions as possible.

Quality of various estimates:

- $|R|$: Relation cardinality. Very accurate but not necessarily perfect (due to concurrent inserts/deletes)
- $|R.x| = |\mathcal{D}_{R.x}|$: Distinct values. Mostly accurate using HyperLogLog sketches
- $|\sigma_p(R)|$: Selection cardinality. Somewhat accurate for simple and not too selective predicates p using random samples
- $|\sigma_p(R).x|$: Distinct values after selection. Inaccurate, no ideal solution

We prefer to use accurate sources of information over inaccurate ones.

We know $|R_1|$, $|R_2|$, domains of $R_1.x$, $R_2.y$, (that is, $|R_1.x|$, $|R_2.y|$), and whether x and y are keys or not.

The selectivity of $\sigma_{R_1.x=c}$ is...

- if x is the key: $\frac{1}{|R_1|}$ (Assume constant is contained within the relation)
- if x is not the key: $\frac{1}{|R_1.x|}$ (Assume values are distributed uniformly)

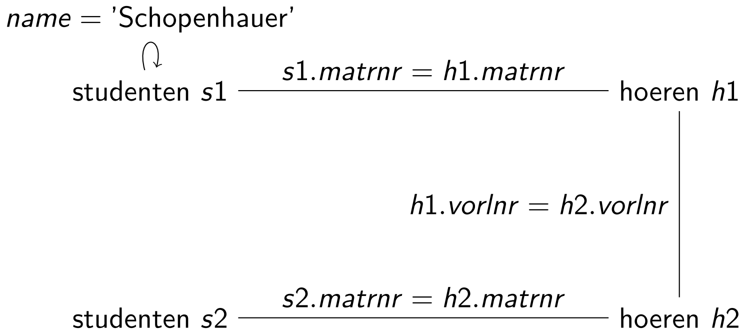
We know $|R_1|$, $|R_2|$, $|R_1.x|$, $|R_2.y|$, and whether x and y are keys or not.
First, the size of $R_1 \times R_2$ is $|R_1||R_2|$
The selectivity of $\bowtie_{R_1.x=R_2.y}$ is...

- if both x and y are the keys: $\frac{1}{max(|R_1|,|R_2|)}$ (Assume larger set is a superset of the smaller set)
- if only x is the key: $\frac{1}{|R_1|}$ (Assume the key attribute is a superset of the other attribute)
- if both x and y are not the keys: $\frac{1}{max(|R_1.x|,|R_2.y|)}$ (Assume values are distributed uniformly)

Homework

Exercise 2

```
select s2.name
from studenten s1, hoeren h1, hoeren h2, studenten s2
where s1.name = 'Schopenhauer' and s1.matrnr = h1.matrnr
  and h1.vorlnr = h2.vorlnr and h2.matrnr = s2.matrnr
```



```
select p.name
from professoren p, vorlesungen v, hoeren h, studenten s
where p.persnr = v.gelesenvon and v.vorlnr = h.vorlnr
  and h.matrnr = s.matrnr and p.name = s.name
```

$$\begin{tikzcd}
p \arrow[r, dash, "p.persnr = v.gelesenVon"] \arrow[d, dash, "p.name = s.name"'] & v \arrow[d, dash, "v.vorlnr = h.vorlnr"] \\
s \arrow[r, dash, "s.matrnr = h.matrnr"] & h
\end{tikzcd}$$

Exercise 3

- Transformative: Move selections down one by one
  - Inefficient
  - Difficult to implement in TinyDB
- Constructive: Build tree bottom-up left-to-right
  - Efficicient
  - Simpler code

Cardinality, Selectivity and Cost Function

- $|\sigma(R_i)| = f_i \cdot |R_i|$
- $|R_1 \bowtie R_2| = f_{1,2} \cdot |R_1||R_2|$
- $|\mathcal{T}| = \begin{cases} f_i \cdot |R_i| & \text{if } \mathcal{T} \text{ is a leaf } R_i \\ (\prod_{R_i \in \mathcal{T}_1, R_j \in \mathcal{T}_2} f_{i,j})|\mathcal{T}_1||\mathcal{T}_2| & \text{if } \mathcal{T} = \mathcal{T}_1 \bowtie \mathcal{T}_2 \text{ (Assume independence of selectivities)} \end{cases}$

Cost functions compare plans. Plans of lower cost are preferred. C_{out} is a simple cost function that sums up intermediate result sizes.

- $C_{\text{out}}(\mathcal{T}) = \begin{cases} 0 & \text{if } \mathcal{T} \text{ is a leaf } R_i \\ |\mathcal{T}| + C_{\text{out}}(\mathcal{T}_1) + C_{\text{out}}(T_2) & \text{if } \mathcal{T} = \mathcal{T}_1 \bowtie \mathcal{T}_2 \end{cases}$

Costs of physical operators:

- $C_{NL}(\mathcal{T}_1 \bowtie \mathcal{T}_2) = |\mathcal{T}_1||\mathcal{T}_2|$
- $C_{HJ}(\mathcal{T}_1 \bowtie \mathcal{T}_2) = 1.2|\mathcal{T}_1|$
- $C_{SMJ}(\mathcal{T}_1 \bowtie \mathcal{T}_2) = |\mathcal{T}_1|log(|\mathcal{T}_1|) + |\mathcal{T}_2|log(|\mathcal{T}_2|)$

Selectivity estimation

We know $|R_1|$, $max(R_1.x)$, $min(R_1.x)$, $R_1.x$ is numeric.

The selectivity of $\sigma_{R_1.x>c}$ is $\frac{max(R_1.x)-c}{max(R_1.x)-min(R_1.x)}$

The selectivity of $\sigma_{c_1<R_1.x<c_2}$ is $\frac{c_2-c_1}{max(R_1.x)-min(R_1.x)}$

Join Ordering

Greedy Heuristics

Linear (left-deep/right-deep/zig-zag) join tree

- GJO-1: Start with smallest relation. Pick next relation with smallest cardinality.
- GJO-2: Start with smallest relation. Pick next relation that minimizes tree cardinality.
- GJO-3: GJO-2, but try every start relation.

Bushy join tree

- GOO: Start with every relation as an individual tree. Pick next pair of trees (using the edges connecting those trees) such that joining them results in the tree with least cardinality.

Repeat until only one tree remains that contains all relations.
Cross-products are disallowed in every algorithm (unless otherwise stated)!

Query Optimization: Exercise

Session 5

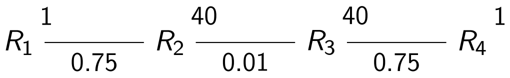
Maximilian Rieger

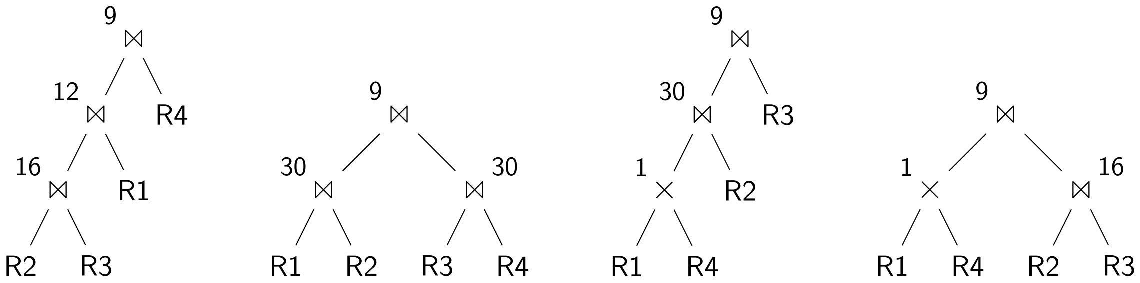
November 24, 2023

Homework

Exercise 1 [cpoptimal]

Give a query graph where the optimal join tree is truly bushy and includes a cross product:





Exercise 2 [greedyalgo]

Give a query graph where GOO does not find the optimal result:

$$\begin{tikzcd}
R_2^{20} \arrow[r, dash, "0.5"'] & R_0^{8} \arrow[r, dash, "0.5"'] & R_1^{20} \arrow[r, dash, "0.2"'] & R_3^{100}
\end{tikzcd}$$

Exercise 4

- Bijective function $f_n\colon \left[0, 2^{n-1}\right) \mapsto \mathcal{T}$
  (left-deep join tree for relations $R_0 - R_1 - \cdots - R_{n-1}$)
- $g_n\colon \left[0, 2^{n-1}\right) \mapsto \{l, r\}^{n-1}$:

$$g_n(x) = g_n(x_1 x_2 \cdots x_{n-1}) = g'(x_1)g'(x_2)\cdots g'(x_{n-1})$$

$$g'(x) = \begin{cases} l, & \text{for } x = 1 \\ r, & \text{for } x = 0 \end{cases}$$

- Bijective function $f_n\colon \left[0, 2^{n-1}\right) \mapsto \mathcal{T}$
  (left-deep join tree for relations $R_0 - R_1 - \cdots - R_{n-1}$)
- $h_n\colon \{l, r\}^{n-1} \mapsto \mathcal{T}$:

$$h_n(x) = h'_n(R_{start(x)}, start(x), start(x), x)$$

$$start(x) = \sum_{x_i \in x} \mathbf{1}_{\{l\}}(x_i)$$

$$h'(S, a, b, x_1 x_2 \cdots x_k) = \begin{cases} S, & \text{for } x = \epsilon \\ h'\left((S \bowtie R_{a-1}), a-1, b, x_2 \cdots x_k\right), & \text{for } x_1 = l \\ h'\left((S \bowtie R_{b+1}), a, b+1, x_2 \cdots x_k\right), & \text{for } x_1 = r \end{cases}$$

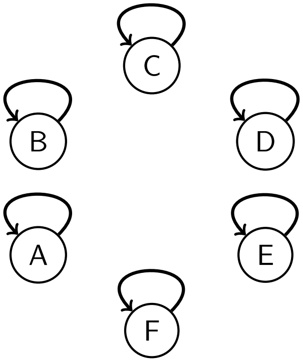
$$R_0 - R_1 - R_2 - R_3$$

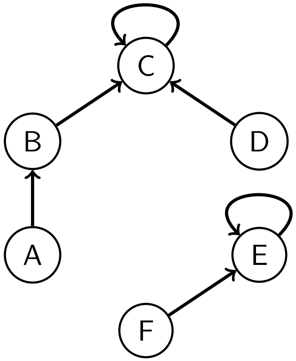
- $f_4(5)$?
- $g_4(5) = g_4(101_2) = lrl$
- $start(lrl) = 1 + 0 + 1 = 2$

$$\begin{aligned}
h_4(lrl) &= h'(R_2,\ 2,\ 2,\ lrl)\\
h'(R_2,\ 2,\ 2,\ lrl) &= h'((R_2 \bowtie R_1),\ 1,\ 2,\ rl)\\
h'((R_2 \bowtie R_1),\ 1,\ 2,\ rl) &= h'(((R_2 \bowtie R_1) \bowtie R_3),\ 1,\ 3,\ l)\\
h'(((R_2 \bowtie R_1) \bowtie R_3),\ 1,\ 3,\ l) &= h'((((R_2 \bowtie R_1) \bowtie R_3) \bowtie R_0),\ 0,\ 3,\ \epsilon)\\
h'((((R_2 \bowtie R_1) \bowtie R_3) \bowtie R_0),\ 0,\ 3,\ \epsilon) &= (((R_2 \bowtie R_1) \bowtie R_3) \bowtie R_0)
\end{aligned}$$

Union Find

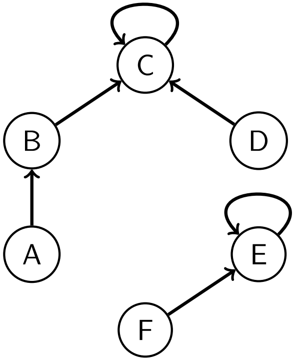
- Manage a family of disjoint sets
- Each set has a representative element
- `find(x)` finds the representative for the set that contains `x`
- `union(a, b)` merges the sets of `a` and `b`

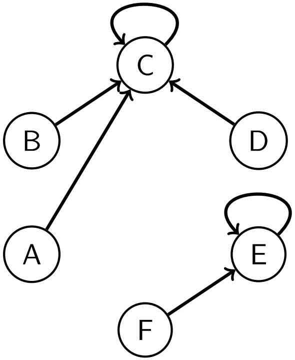




Path Compression

- A naive implementation would create longer and longer chains
- $\Rightarrow$ for every element visited in `find(x)`, make it point directly to the representative
- `union(a, b)` calls `find(a)` and `find(b)`!





```
Find(x):
  repr := x
  while repr.parent != repr:
    repr := repr.parent

  while x.parent != repr:
    parent := x.parent
    x.parent := repr
    x := parent

  return repr
```

```
Union(a, b):
  a := Find(a)
  b := Find(b)

  if a = b:
    return

  if a.rank < b.rank:
    a.parent = b

  else if b.rank < a.rank:
    b.parent = a

  else: //a.rank = b.rank
    a.parent := b
    b.rank++
```

IKKBZ

- Query graph Q is acyclic.
- Pick a root node, turn it into a tree.
- Run the following procedure for every root node
- Select the cheapest plan

Input: Precedence graph rooted in some node

1. if the tree is a single chain, stop
2. find the subtree (rooted at r) all of whose children are chains
3. normalize, if $c_1 \rightarrow c_2$, but $rank(c_1) > rank(c_2)$ in the subtree rooted at r
4. merge chains in the subtree rooted at r, rank is ascending
5. repeat 1

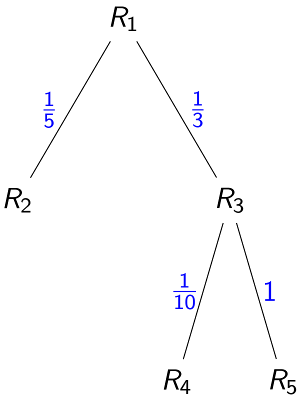
For every relation R_i we keep

- cardinality n_i
- selectivity s_i — the selectivity of the incoming edge from the parent of R_i
- cost $C(R_i) = n_i s_i$ (or 0, if R_i is the root)
- rank $r_i = \frac{T(r_i)-1}{C(r_i)} = \frac{n_i s_i - 1}{n_i s_i}$

Moreover,

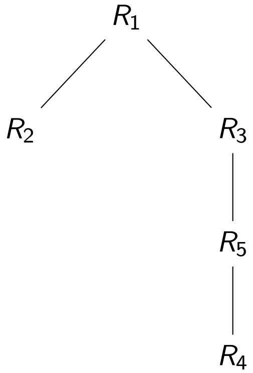
- $C(S_1 S_2) = C(S_1) + T(S_1)C(S_2)$ ($\sim$Cost)
- $T(S) = \prod_{R_i \in S}(s_i n_i)$ ($\sim$Cardinality)
- rank of a sequence $r(S) = \frac{T(S)-1}{C(S)}$

- what is the rank?
- increase of the intermediate result size normalized by the cost of doing the join
- when is $(R_1 \bowtie R_2) \bowtie R_3$ cheaper than $(R_1 \bowtie R_3) \bowtie R_2$?
- if $r(R_2) < r(R_3)$!



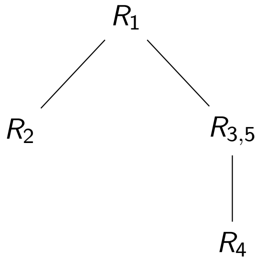
Relation	n	s	C	T	*rank*
2	20	$\frac{1}{5}$	4	4	$\frac{3}{4}$
3	30	$\frac{1}{3}$	10	10	$\frac{9}{10}$
4	50	$\frac{1}{10}$	5	5	$\frac{4}{5}$
5	2	1	2	2	$\frac{1}{2}$

Subtree R_3: merging, $rank(R_5) < rank(R_4)$

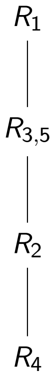


Relation	n	s	C	T	*rank*
2	20	$\frac{1}{5}$	4	4	$\frac{3}{4}$
3	30	$\frac{1}{3}$	10	10	$\frac{9}{10}$
4	50	$\frac{1}{10}$	5	5	$\frac{4}{5}$
5	2	1	2	2	$\frac{1}{2}$

Subtree R_1: $rank(R_3) > rank(R_5)$, normalizing

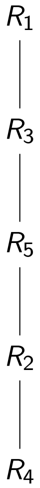


Relation	n	s	C	T	*rank*
2	20	$\frac{1}{5}$	4	4	$\frac{3}{4}$
3	30	$\frac{1}{3}$	10	10	$\frac{9}{10}$
4	50	$\frac{1}{10}$	5	5	$\frac{4}{5}$
5	2	1	2	2	$\frac{1}{2}$
3,5			30	20	$\frac{19}{30}$



Relation	n	s	C	T	*rank*
2	20	$\frac{1}{5}$	4	4	$\frac{3}{4}$
3	30	$\frac{1}{15}$	10	10	$\frac{9}{10}$
4	50	$\frac{1}{10}$	5	5	$\frac{4}{5}$
5	2	1	2	2	$\frac{1}{2}$
3,5			30	20	$\frac{19}{30}$

Subtree R_1: merging

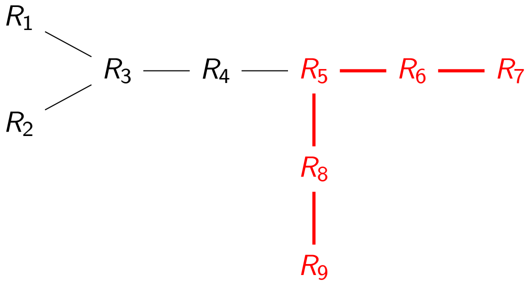


Relation	n	s	C	T	*rank*
2	20	$\frac{1}{5}$	4	4	$\frac{3}{4}$
3	30	$\frac{1}{15}$	10	10	$\frac{9}{10}$
4	50	$\frac{1}{10}$	5	5	$\frac{4}{5}$
5	2	1	2	2	$\frac{1}{2}$
3,5			30	20	$\frac{3}{30}$

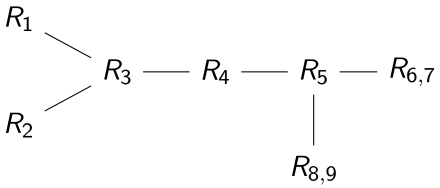
Denormalizing

$$\begin{tikzcd}
R_1 \arrow[dr, dash, "\frac{1}{6}"] & & & & & \\
& R_3 \arrow[r, dash, "\frac{1}{20}"] & R_4 \arrow[r, dash, "\frac{3}{4}"] & R_5 \arrow[r, dash, "\frac{1}{2}"] \arrow[d, dash, "\frac{1}{5}"] & R_6 \arrow[r, dash, "\frac{1}{14}"] & R_7 \\
R_2 \arrow[ur, dash, "\frac{1}{10}"'] & & & R_8 \arrow[d, dash, "\frac{1}{25}"] & & \\
& & & R_9 & &
\end{tikzcd}$$

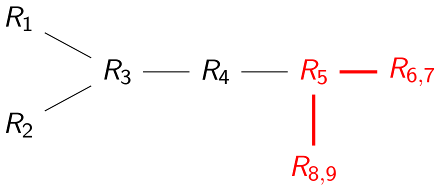
- $|R_1| = 30$
- $|R_2| = 100$
- $|R_3| = 30$
- $|R_4| = 20$
- $|R_5| = 10$
- $|R_6| = 20$
- $|R_7| = 70$
- $|R_8| = 100$
- $|R_9| = 100$



- $r(R_2) = \frac{9}{10} = 0.9$
- $r(R_3) = \frac{4}{5} = 0.8$
- $r(R_4) = 0$
- $r(R_5) = \frac{13}{15} \approx 0.86$
- $r(R_6) = \frac{9}{10} = 0.9$
- $r(R_7) = \frac{4}{5} = 0.8$
- $r(R_8) = \frac{19}{20} = 0.95$
- $r(R_9) = \frac{3}{4} = 0.75$



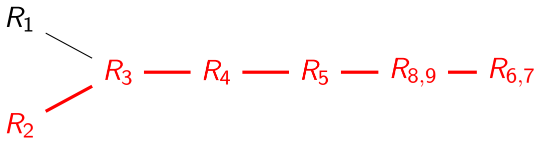
- $C(R_{8,9}) = 100$
- $T(R_{8,9}) = 80$
- $r(R_{8,9}) = \frac{79}{100} = 0.79$
- $C(R_{6,7}) = 60$
- $T(R_{6,7}) = 50$
- $r(R_{6,7}) = \frac{49}{60} \approx 0.816$



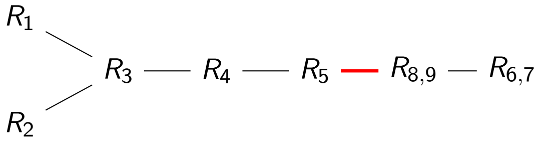
- $C(R_{8,9}) = 100$
- $T(R_{8,9}) = 80$
- $r(R_{8,9}) = \frac{79}{100} = 0.79$
- $C(R_{6,7}) = 60$
- $T(R_{6,7}) = 50$
- $r(R_{6,7}) = \frac{49}{60} \approx 0.816$

$$\begin{tikzcd}
R_1 \arrow[dr, dash] & & & & & \\
& R_3 \arrow[r, dash] & R_4 \arrow[r, dash] & R_5 \arrow[r, dash] & R_{8,9} \arrow[r, dash] & R_{6,7} \\
R_2 \arrow[ur, dash] & & & & &
\end{tikzcd}$$

- $r(R_{8,9}) < r(R_{6,7})$



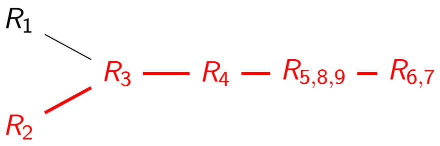
- $r(R_4) = 0$
- $r(R_5) = \frac{13}{15} \approx 0.86$
- $r(R_{8,9}) = \frac{79}{100} = 0.79$
- $r(R_{6,7}) = \frac{49}{60} \approx 0.81$



- $r(R_5) = \frac{13}{15} \approx 0.86$
- $r(R_{8,9}) = 0.79$

$$\begin{tikzcd}
R_1 \arrow[dr, dash] & & & & \\
 & R_3 \arrow[r, dash] & R_4 \arrow[r, dash] & R_{5,8,9} \arrow[r, dash] & R_{6,7} \\
R_2 \arrow[ur, dash] & & & &
\end{tikzcd}$$

- $C_{5,8,9} = \frac{1515}{2}$
- $T_{5,8,9} = 600$
- $r(R_{5,8,9}) = \frac{1198}{1515} \approx 0.79$
- $r(R_{6,7}) \approx 0.816$



- $r(R_2) = \frac{9}{10}$
- $r(R_3) = 0.8$
- $r(R_4) = 0$
- $r(R_{5,8,9}) = \frac{1198}{1515} \approx 0.79$
- $r(R_{6,7}) \approx 0.816$

$$R_1 - R_3 - R_4 - R_{5,8,9} - R_{6,7} - R_2$$

$$R_1 \text{ --- } R_3 \text{ --- } R_4 \text{ --- } R_5 \text{ --- } R_8 \text{ --- } R_9 \text{ --- } R_6 \text{ --- } R_7 \text{ --- } R_2$$

IKKBZ-based heuristics

What if Q has cycles?

- Observation 1: the answer of the query, corresponding to any subgraph of the query graph, is a superset of the answer to the original query
- Observation 2: a very selective join is more likely to be influential in choosing the order than a non-selective join

Build the minimum spanning tree (minimize the product of the edge weights), compute the total order, compute the original query.

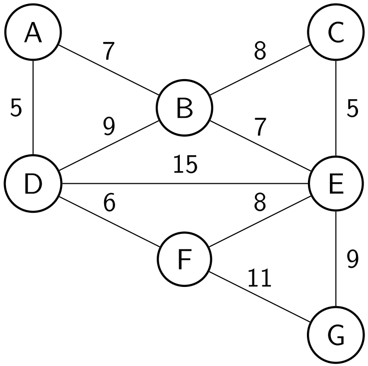
Minimal Spanning Trees (MST)

- Why do we need this?
- What algorithms do you know?
  - Kruskal
  - Prim

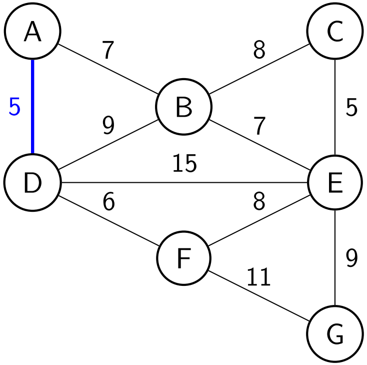
Kruskal

- Sort all edges by weight
- Iterate starting with smallest weighted edge
- If components of edge are not connected yet, add edge to MST
- Can use union find to keep track of connected components

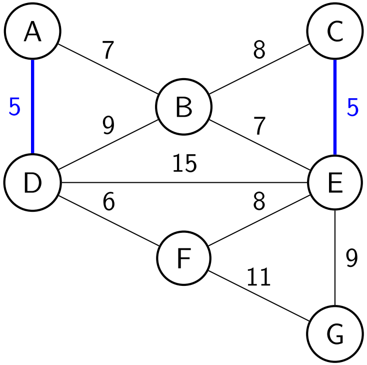
- AD 5 ●
- CE 5
- DF 6
- AB 7
- BE 7
- BC 8
- EF 8
- BD 9
- EG 9
- FG 11
- DE 15



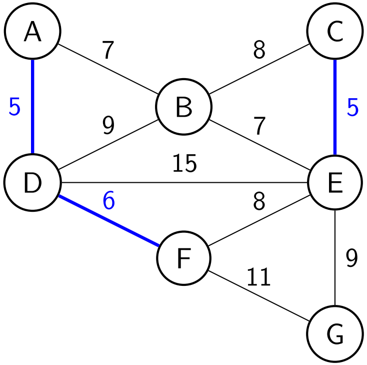
- AD 5
- CE 5 ●
- DF 6
- AB 7
- BE 7
- BC 8
- EF 8
- BD 9
- EG 9
- FG 11
- DE 15



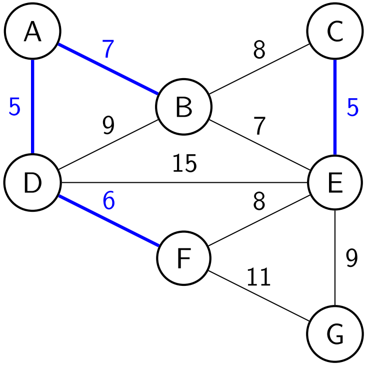
- AD 5
- CE 5
- DF 6 ●
- AB 7
- BE 7
- BC 8
- EF 8
- BD 9
- EG 9
- FG 11
- DE 15



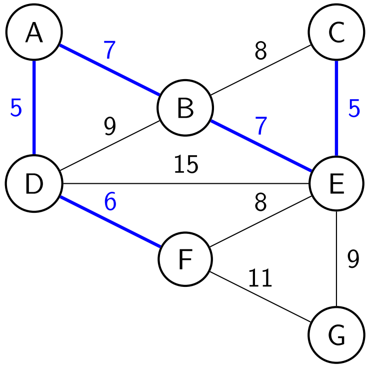
- AD 5
- CE 5
- DF 6
- AB 7 ●
- BE 7
- BC 8
- EF 8
- BD 9
- EG 9
- FG 11
- DE 15



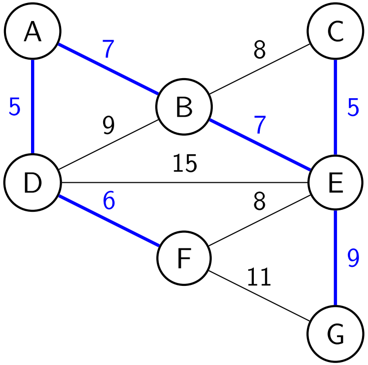
- AD 5
- CE 5
- DF 6
- AB 7
- BE 7 ●
- BC 8
- EF 8
- BD 9
- EG 9
- FG 11
- DE 15



- AD 5
- CE 5
- DF 6
- AB 7
- BE 7
- BC 8 ●
- EF 8
- BD 9
- EG 9
- FG 11
- DE 15



- AD 5
- CE 5
- DF 6
- AB 7
- BE 7
- BC 8
- EF 8
- BD 9
- EG 9
- FG 11 ●
- DE 15

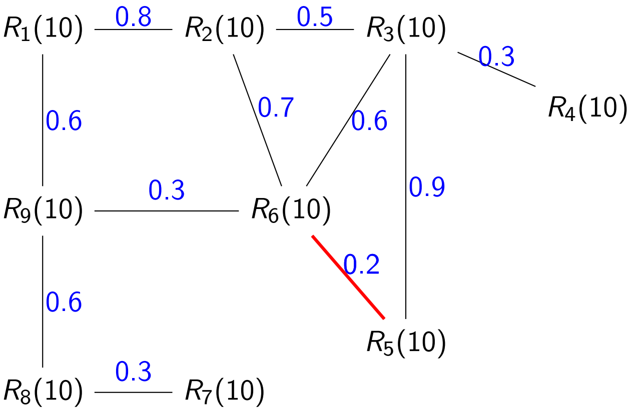


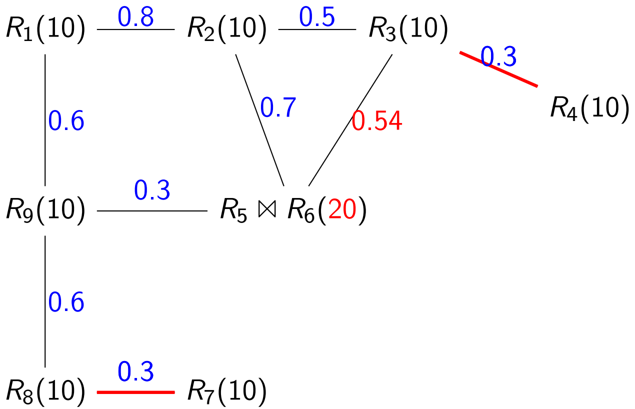
Greedy Join Ordering

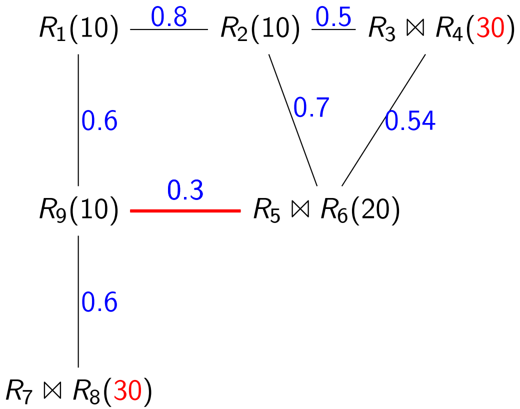
Greedy Operator Ordering (GOO)

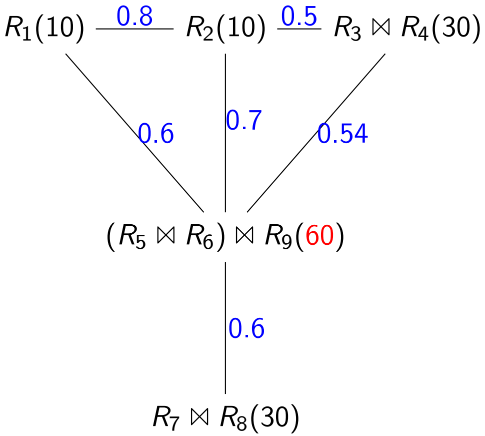
- take the query graph
- find relations R_1, R_2 such that $|R_1 \bowtie R_2|$ is minimal and merge them into one node
- repeat as long as the query graph has more than one node

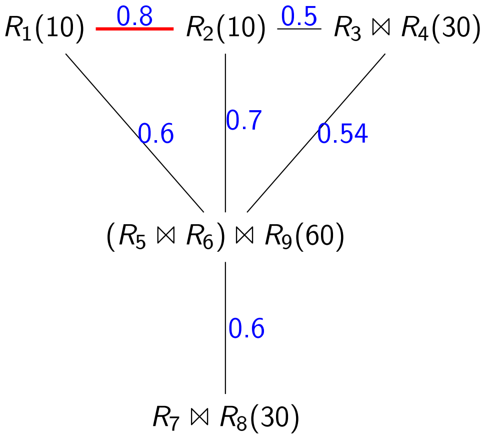
Generates bushy trees!


$R_1(10)$
0.8
$R_2(10)$
0.5
$R_3(10)$
0.3
$R_4(10)$
0.6
0.7
0.6
0.9
$R_9(10)$
0.3
$R_6(10)$
0.2
0.6
$R_5(10)$
$R_8(10)$
0.3
$R_7(10)$

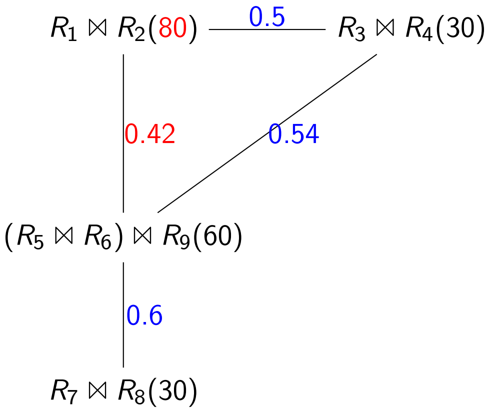

$R_1(10)$
0.8
$R_2(10)$
0.5
$R_3(10)$
0.3
$R_4(10)$
0.6
0.7
0.54
$R_9(10)$
0.3
$R_5 \bowtie R_6(20)$
0.6
$R_8(10)$
0.3
$R_7(10)$

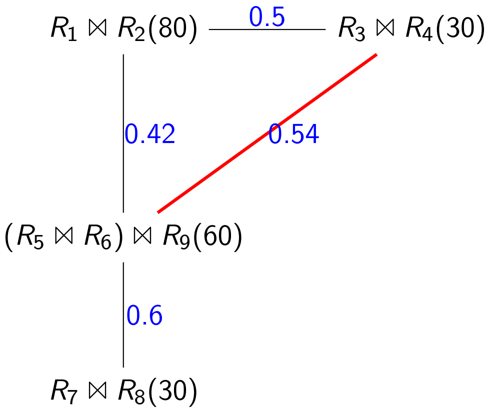

$R_1(10)$
0.8
$R_2(10)$
0.5
$R_3 \bowtie R_4(30)$
0.6
0.7
0.54
$R_9(10)$
0.3
$R_5 \bowtie R_6(20)$
0.6
$R_7 \bowtie R_8(30)$


$R_1(10)$
0.8
$R_2(10)$
0.5
$R_3 \bowtie R_4(30)$
0.6
0.7
0.54
$(R_5 \bowtie R_6) \bowtie R_9(60)$
0.6
$R_7 \bowtie R_8(30)$

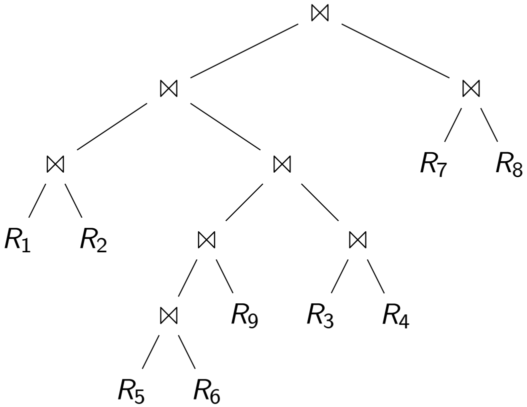


$R_1(10)$
0.8
$R_2(10)$
0.5
$R_3 \bowtie R_4(30)$
0.6
0.7
0.54
$(R_5 \bowtie R_6) \bowtie R_9(60)$
0.6
$R_7 \bowtie R_8(30)$


$R_1 \bowtie R_2(80)$
0.5
$R_3 \bowtie R_4(30)$
0.42
0.54
$(R_5 \bowtie R_6) \bowtie R_9(60)$
0.6
$R_7 \bowtie R_8(30)$


$R_1 \bowtie R_2 (80)$
0.5
$R_3 \bowtie R_4 (30)$
0.42
0.54
$(R_5 \bowtie R_6) \bowtie R_9 (60)$
0.6
$R_7 \bowtie R_8 (30)$

$$\begin{tikzcd}
R_1 \bowtie R_2 (80) \arrow[d, dash, "0.21"] \\
((R_5 \bowtie R_6) \bowtie R_9) \bowtie (R_3 \bowtie R_4) (972) \arrow[d, dash, "0.6"] \\
R_7 \bowtie R_8 (30)
\end{tikzcd}$$


$\bowtie$
$\bowtie$
$\bowtie$
$\bowtie$
$\bowtie$
R_7
R_8
R_1
R_2
$\bowtie$
$\bowtie$
$\bowtie$
R_9
R_3
R_4
R_5
R_6

Query Optimization: Exercise

Session 7

Maximilian Rieger

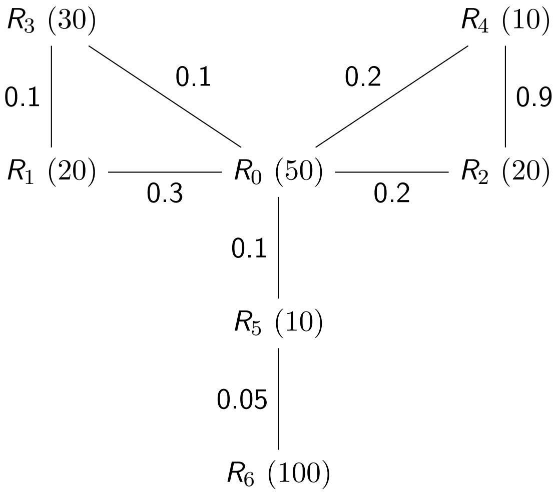
December 8, 2023

Exercise [ikkbzmvp]

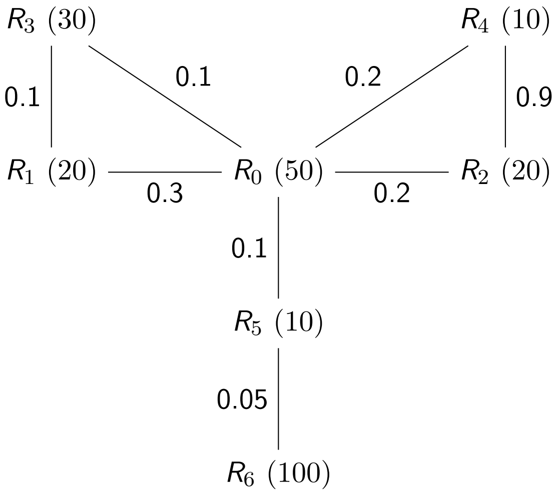
$$\begin{tikzcd} R_0^{10} \arrow[r, dash, "0.1"'] & R_1^{20} \arrow[r, dash, "0.2"'] & R_2^{10} \arrow[r, dash, "0.05"'] & R_3^{100} \end{tikzcd}$$

- Root in R_0 trivial: $((R_0 \bowtie R_1) \bowtie R_2) \bowtie R_3$
- Root in R_1
  - $n_i s_i$: R_0:1, R_2:2, R_3:5
  - no rank needed! $((R_1 \bowtie R_0) \bowtie R_2) \bowtie R_3$
- Root in R_2
  - $n_i s_i$: R_0:1, R_1:4, R_3:5
  - normalize R_1 and R_0
    - $C(S_1 S_2) = C(S_1) + T(S_1)C(S_2)$ ($\sim$Cost)
    - $T(S) = \prod_{R_i \in S}(s_i n_i)$ ($\sim$Cardinality)
    - rank of a sequence $r(S) = \frac{T(S)-1}{C(S)}$
  - $r(R_1 R_0) = 3/8$, $r(R3) = 4/5$ $((R_2 \bowtie R_1) \bowtie R_0) \bowtie R_3$
- Root in R_3 trivial $((R_3 \bowtie R_2) \bowtie R_1) \bowtie R_0$
- Can MVP find a better tree? $((R_0 \bowtie R_1) \bowtie R_2) \bowtie R_3$ is optimal

Exercise [mst]



Exercise [mst]



- R5 R6 0.05
- R0 R3 0.1
- R0 R5 0.1
- R1 R3 0.1
- R0 R2 0.2
- R0 R4 0.2
- R0 R1 0.3
- R0 R1 0.9

DP Table

- different DP algorithms have different enumeration
- all store optimal subtrees in the dp table
- an entry maps from a set of relations to its optimal tree
- by the time an entry is used, it needs to be optimal
  enumeration order is important

Exercise [dpsizedpsub]

DPsize:

$$\begin{tikzcd}
R_0^{20} \arrow[r, dash, "0.2"'] \arrow[d, dash, "0.2"'] & R_1^{10} \arrow[r, dash, "0.4"'] \arrow[d, dash, "0.2"'] & R_4^{10} \\
R_3^{20} \arrow[r, dash, "0.4"'] & R_2^{20} &
\end{tikzcd}$$

size 1:

- $\{R0\}$
- $\{R1\}$
- $\{R2\}$
- $\{R3\}$
- $\{R4\}$

Size 2:

- $\{R0, R1\}$
- $\{R0, R3\}$
- $\{R1, R2\}$
- $\{R1, R4\}$
- $\{R2, R3\}$

Size 3:

- $\{R0, R1, R2\}$
- $\{R0, R1, R3\}$
- $\{R0, R1, R4\}$
- $\{R0, R2, R3\}$
- $\{R1, R2, R3\}$
- $\{R1, R2, R4\}$

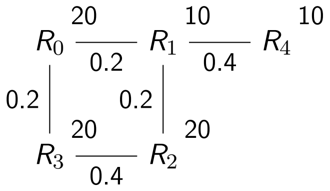
Size 4:

- $\{R0, R1, R2, R3\}$
- $\{R0, R1, R2, R4\}$
- $\{R0, R1, R3, R4\}$
- $\{R1, R2, R3, R4\}$

Size 5:

- $\{R0, R1, R2, R3, R4\}$

DPsub:



```
 1:     1: R0
 2:    10: R1
 3:    11: R0R1
  R0 - R1
 4:   100: R2
 5:   101: R0R2
  R0 - R2
 6:   110: R1R2
  R1 - R2
 7:   111: R0R1R2
  R0 - R1R2
  R1 - R0R2
  R0R1 - R2
 8:  1000: R3
 9:  1001: R0R3
  R0 - R3
10:  1010: R1R3
  R1 - R3
11:  1011: R0R1R3
  R0 - R1R3
  R1 - R0R3
  R0R1 - R3
12:  1100: R2R3
  R2 - R3
13:  1101: R0R2R3
  R0 - R2R3
  R2 - R0R3
  R0R2 - R3
14:  1110: R1R2R3
  R1 - R2R3
  R2 - R1R3
  R1R2 - R3
15:  1111: R0R1R2R3
  R0 - R1R2R3
  R1 - R0R2R3
  R0R1 - R2R3
  R2 - R0R1R3
  R0R2 - R1R3
  R1R2 - R0R3
  R0R1R2 - R3
16: 10000: R4
...
```

Algorithms

so far:

- GreedyJoinOrdering-{1,2,3} heuristics
- Greedy Operator Ordering (GOO) [1]
- IKKBZ [2] [3]
- Maximum Value Precedence (MVP) [4]
- DPsize, DPsub

today:

- DPccp
- Hypergraphs
- Simplifying the query graph

DPccp

- standard DP algorithms consider many pairs that are discarded due to disconnectedness or failing disjointness tests
- taking the query graph into account avoids this [6]

- EnumerateCsg+EnumerateCmp produce all ccp
- resulting algorithm DPccp considers exactly #ccp pairs
- which is the lower bound for all DP enumeration algorithms

```
DPccp($R$)
Input:   a connected query graph with relations $R = \{R_0, \ldots, R_{n-1}\}$
Output: an optimal bushy join tree
$B$ = an empty DP table $2^R \rightarrow$ join tree
for $\forall R_i \in R$
   $B[\{R_i\}] = R_i$
for $\forall$ csg-cmp-pairs $(S_1, S_2)$, $S = S_1 \cup S_2$ {
   $p_1 = B[S_1]$, $p_2 = B[S_2]$
   $P$ = CreateJoinTree$(p_1, p_2)$;
   if $B[S] = \epsilon \vee C(B[S]) > C(P)$
      $B[S] = P$
}
return $B[\{R_0, \ldots, R_{n-1}\}]$
```

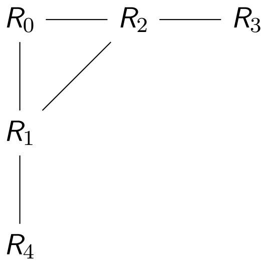
```
EnumerateCsg($G$):
    EnumerateCsg($G$, $\{n-1, ..., 0\}$, $\emptyset$);

EnumerateCsg($G$, $S$, $X$):
    **for all** $i \in S$ **descending** {
        EnumerateCsgRec($G$, $\{v_i\}$, $X \cup (\mathcal{B}_i \cap S)$);
    }

EnumerateCsgRec($G$, $S$, $X$):
    **emit** $(S)$;
    $N = \mathcal{N}(S) \setminus X$;
    **for all** $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
        EnumerateCsgRec($G$, $(S \cup S')$, $(X \cup N)$);
    }
```

```
EnumerateCmp($G$,$S_1$):
    $X = \mathcal{B}_{\min(S_1)} \cup S_1$;
    $N = \mathcal{N}(S_1) \setminus X$;
    EnumerateCsg($G$, $N$, $X$);
}
```

Example



```
{4} - X
{3} - X
{2} - {3}
{2, 3} - X
{1} - {4}, {2}, {2, 3}
{1, 2} - {4}, {3}
{1, 2, 3} - {4}
{1, 4} - {2}, {2, 3}
{1, 2, 4} - {3}
{1, 2, 3, 4} - X
{0} - {2}, {2, 3}, {1}, {1, 2}, {1, 2, 3}, {1, 4},
      {1, 2, 4}, {1, 2, 3, 4}
{0, 1} - {4}, {2}, {2, 3}
{0, 1, 4} - {2}, {2, 3}
{0, 2} - {3}, {1}, {1, 4}
{0, 2, 3} - {1}, {1, 4}
{0, 1, 2} - {4}, {3}
{0, 1, 2, 3} - {4}
{0, 1, 2, 4} - {3}
{0, 1, 2, 3, 4} - X
```

Hypergraphs

- Complex Join Predicates
- Reordering constraints due to non-inner Joins
- Hyperedges connect **two** sets of nodes: (S_1, S_2), e.g. $(\{R_0, R_1\}, \{R_2\})$
- $S_1 \cap S_2 = \emptyset$

DPhyp [7]:

- Similar to DPccp but hyperedges 'lead' to the smallest node within the set
- Subproblems may be disconnected. Must check for connectedness!

Query Simplification

Simplify the query graph if it is too complex.

- GOO: greedily choose joins to perform
- Simplification: greedily choose joins that must be avoided (we can start with 'obvious' decisions)
- benefit$(X \bowtie_1 R_1, X \bowtie_2 R_2) = \frac{C((X \bowtie_1 R_1) \bowtie_2 R_2)}{C((X \bowtie_2 R_2) \bowtie_1 R_1)}$
- If benefit$(X \bowtie_1 R_1, X \bowtie_2 R_2)$ is high, we want to perform $\bowtie_2$ before $\bowtie_1$
- pick the pair with highest benefit
- prefer $\bowtie_2$ over $\bowtie_1$, i.e. replace the edge $(\{X\}, \{R_1\})$ with a hyperedge $(\{X, R_2\}, \{R_1\})$
- Important: consider all possible edge combinations, that is, benefit$(R_0 \bowtie R_1, R_0 \bowtie R_2)$ as well as *benefit*$(R_0 \bowtie R_2, R_0 \bowtie R_1)$

Homework

- star queries are special
- you can use tinydb with real (fake) data!
- do DPccp
- do query simplification

Query Optimization: Exercise

Session 8

Altan Birler

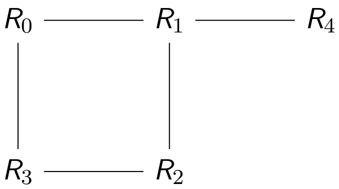
February 5, 2024

Exercise 1

Proof sketch by reduction to IKKBZ:

- All join trees of a star query graph are linear.
- IKKBZ finds optimal linear plans for acyclic graphs.
- IKKBZ finds optimal plans for star query graphs.
- $r(R_i) < r(R_j) \iff s_i|R_i| < s_j|R_j|$
- GreedyStar computes the same tree as IKKBZ.
- GreedyStar is optimal for star query graphs.

Exercise 3


$$\begin{tikzcd} R_0 \arrow[r, dash] \arrow[d, dash] & R_1 \arrow[r, dash] \arrow[d, dash] & R_4 \\ R_3 \arrow[r, dash] & R_2 & \end{tikzcd}$$

CSG	CMP
$\{4\}$	$\emptyset$
$\{3\}$	$\emptyset$
$\{2\}$	$\{3\}$
$\{2,3\}$	$\emptyset$
$\{1\}$	$\{4\}$ $\{2\}$ $\{2,3\}$
$\{1,2\}$	$\{4\}$ $\{3\}$
$\{1,2,3\}$	$\{4\}$
$\{1,4\}$	$\{2\}$ $\{2,3\}$
$\{1,2,4\}$	$\{3\}$
$\{1,2,3,4\}$	$\emptyset$
$\{0\}$	$\{3\}$ $\{2,3\}$ $\{1\}$ $\{1,2\}$ $\{1,2,3\}$ $\{1,4\}$ $\{1,2,4\}$ $\{1,2,3,4\}$
$\{0,1\}$	$\{4\}$ $\{3\}$ $\{2\}$ $\{2,3\}$
$\{0,1,2\}$	$\{4\}$ $\{3\}$
$\{0,1,4\}$	$\{3\}$ $\{2\}$ $\{2,3\}$
$\{0,1,2,4\}$	$\{3\}$
$\{0,3\}$	$\{2\}$ $\{1\}$ $\{1,2\}$ $\{1,4\}$ $\{1,2,4\}$
$\{0,2,3\}$	$\{1\}$ $\{1,4\}$
$\{0,1,3\}$	$\{4\}$ $\{2\}$
$\{0,1,2,3\}$	$\{4\}$
$\{0,1,3,4\}$	$\{2\}$
$\{0,1,2,3,4\}$	$\emptyset$

Exercise [simplification]

$$\text{benefit}(X \bowtie_1 R_1, X \bowtie_2 R_2) = \frac{C_{out}((X \bowtie_1 R_1) \bowtie_2 R_2)}{C_{out}((X \bowtie_2 R_2) \bowtie_1 R_1)}$$

Exercise [simplification]

$$\text{benefit}(X \bowtie_1 R_1, X \bowtie_2 R_2) = \frac{C_{out}((X \bowtie_1 R_1) \bowtie_2 R_2)}{C_{out}((X \bowtie_2 R_2) \bowtie_1 R_1)}$$

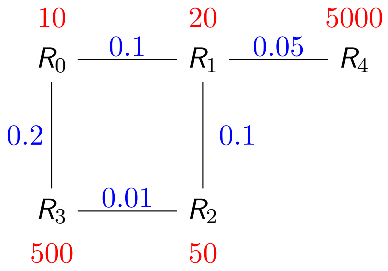
If $\text{benefit}(X \bowtie_1 R_1, X \bowtie_2 R_2)$ is high, we want to perform $\bowtie_2$ before $\bowtie_1$

Exercise [simplification]

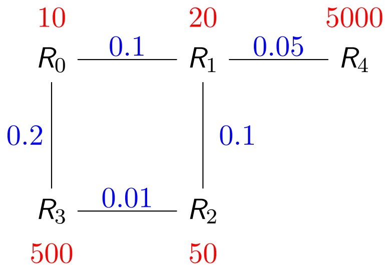
$$\text{benefit}(X \bowtie_1 R_1, X \bowtie_2 R_2) = \frac{C_{out}((X \bowtie_1 R_1) \bowtie_2 R_2)}{C_{out}((X \bowtie_2 R_2) \bowtie_1 R_1)}$$

If $\text{benefit}(X \bowtie_1 R_1, X \bowtie_2 R_2)$ is high, we want to perform $\bowtie_2$ before $\bowtie_1$

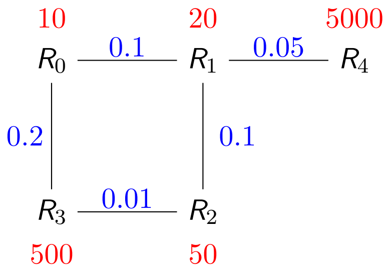
Important: consider all possible edge combinations, that is, $\mathit{benefit}(R_0 \bowtie R_1, R_0 \bowtie R_2)$ as well as $\mathit{benefit}(R_0 \bowtie R_2, R_0 \bowtie R_1)$



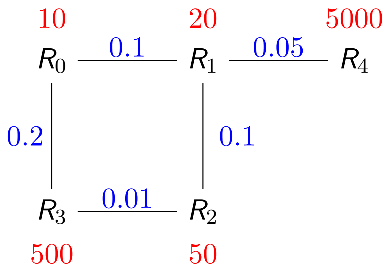
- benefit$(R_0 \bowtie R_1, R_0 \bowtie R_3) = \frac{202}{300}$
- $b(R_0 \bowtie R_3, R_0 \bowtie R_1) = 300/202$
- $b(R_1 \bowtie R_2, R_1 \bowtie R_0) = 20/12$
- $b(R_3 \bowtie R_0, R_3 \bowtie R_2) = 2$
- $b(R_2 \bowtie R_3, R_2 \bowtie R_1) = 5/4$
- $b(R_1 \bowtie R_4, R_1 \bowtie R_0) = 500/251$
- $b(R_1 \bowtie R_4, R_1 \bowtie R_2) = 300/251$



- benefit$(R_0 \bowtie R_1, R_0 \bowtie R_3) = \frac{202}{300}$
- $b(R_0 \bowtie R_3, R_0 \bowtie R_1) = 300/202$
- $b(R_1 \bowtie R_2, R_1 \bowtie R_0) = 20/12$
- $b(R_3 \bowtie R_0, R_3 \bowtie R_2) = 2$
- $b(R_2 \bowtie R_3, R_2 \bowtie R_1) = 5/4$
- $b(R_1 \bowtie R_4, R_1 \bowtie R_0) = 500/251$
- $b(R_1 \bowtie R_4, R_1 \bowtie R_2) = 300/251$
- $R_3 \bowtie R_2$ before $R_3 \bowtie R_0$.
  Replace $R_0 - R_3$ by $\{R_0\} - \{R_2, R_3\}$



- $\text{benefit}(R_0 \bowtie R_1, R_0 \bowtie R_3) = \frac{202}{300}$
- $b(R_0 \bowtie R_3, R_0 \bowtie R_1) = 300/202$
- $b(R_1 \bowtie R_2, R_1 \bowtie R_0) = 20/12$
- $b(R_2 \bowtie R_3, R_2 \bowtie R_1) = 5/4$
- $b(R_1 \bowtie R_4, R_1 \bowtie R_0) = 500/251$
- $b(R_1 \bowtie R_4, R_1 \bowtie R_2) = 300/251$
- $b(R_0 \bowtie (R_3 \bowtie R_2), R_0 \bowtie R_1) = \frac{C((R_0 \bowtie (R_3 \bowtie R_2)) \bowtie R_1)}{C((R_0 \bowtie R_1) \bowtie (R_3 \bowtie R_2))} = 850/370$
- $b((R_2 \bowtie R_3) \bowtie R_0, R_2 \bowtie R_1) = \frac{C(((R_2 \bowtie R_3) \bowtie R_0) \bowtie R_1)}{C(((R_2 \bowtie R_3) \bowtie R_1) \bowtie R_0)} = 1$



- benefit$(R_0 \bowtie R_1, R_0 \bowtie R_3) = \frac{202}{300}$
- $b(R_0 \bowtie R_3, R_0 \bowtie R_1) = 300/202$
- $b(R_1 \bowtie R_2, R_1 \bowtie R_0) = 20/12$
- $b(R_2 \bowtie R_3, R_2 \bowtie R_1) = 5/4$
- $b(R_1 \bowtie R_4, R_1 \bowtie R_0) = 500/251$
- $b(R_1 \bowtie R_4, R_1 \bowtie R_2) = 300/251$
- $b(R_0 \bowtie (R_3 \bowtie R_2), R_0 \bowtie R_1) = \frac{C((R_0 \bowtie (R_3 \bowtie R_2)) \bowtie R_1)}{C((R_0 \bowtie R_1) \bowtie (R_3 \bowtie R_2))} = 850/370$
- $b((R_2 \bowtie R_3) \bowtie R_0, R_2 \bowtie R_1) = \frac{C(((R_2 \bowtie R_3) \bowtie R_0) \bowtie R_1)}{C(((R_2 \bowtie R_3) \bowtie R_1) \bowtie R_0)} = 1$
- $R_0 \bowtie R_1$ before $R_0 \bowtie (R_3 \bowtie R_2)$.
  Replace $\{R_0\} - \{R_2, R_3\}$ by $\{R_0, R_1\} - \{R_2, R_3\}$

Adaptive Optimization

- huge spread in query complexities
- different expectations
  - small/easy queries: solve optimally (DPhyp)
  - medium queries: generate decent plans but keep optimization fast (search space linearization)
  - large queries: gracefully introduce greediness (IDP; to be discussed in the lecture)
- how to measure query complexity?

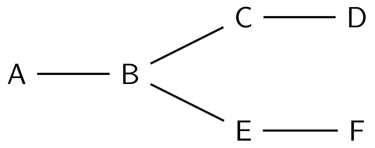
Search Space Linearization

- can find the optimal bushy plan given the proper permutation of relations in $\mathcal{O}(n^3)$
- permutation of relations in the optimal plan unknown!
- use the solutions of IKKBZ as seed for the linearized DP (heuristic)

Linearized DP

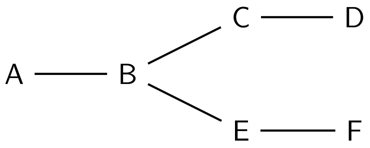
```
LinDP($R$)
Input:   a sequence of relations $R = (R_0, \ldots, R_{n-1})$
Output:  an optimal bushy join tree (given the order)
$B$ = an empty DP table $R^3 \rightarrow$ join tree
for $\forall R_i \in R$
  $B[R_i] = R_i$
for each $2 \leq l \leq n$
  for each $0 \leq s \leq n - l$
    for each $1 \leq k \leq l - 1$
      if $\neg$connected $(R_s, \ldots, R_{s+k-1}), (R_{s+k}, \ldots, R_{s+l-1})$
        continue
      $P$ = CreateJoinTree($B[R_s, \ldots, R_{s+k-1}], B[R_{s+k}, \ldots, R_{s+l-1}]$);
      if $B[R_s, \ldots, R_{s+l-1}] = \epsilon \vee C(B[R_s, \ldots, R_{s+l-1}]) > C(P)$
        $B[R_s, \ldots, R_{s+l-1}] = P$
}
return $B[R_0, \ldots, R_{n-1}]$
```

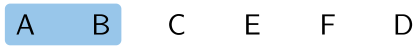
- Optimally combine optimal solutions for subchains



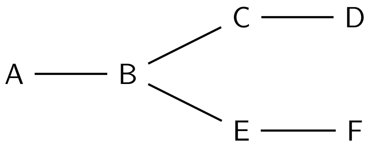
A B C E F D

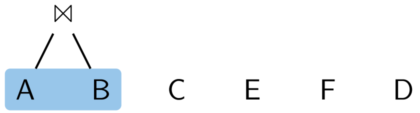
- Optimally combine optimal solutions for subchains



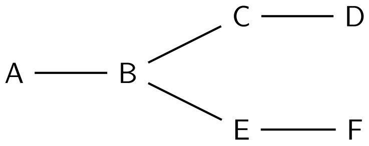


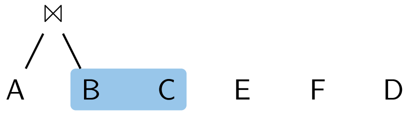
- Optimally combine optimal solutions for subchains



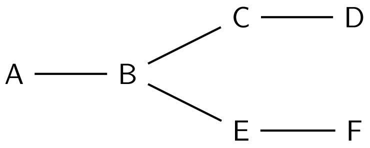


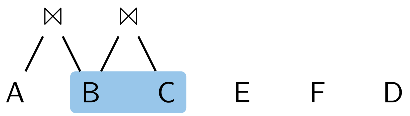
- Optimally combine optimal solutions for subchains



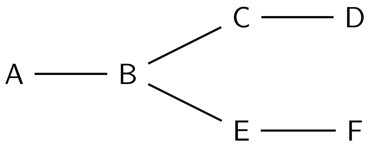


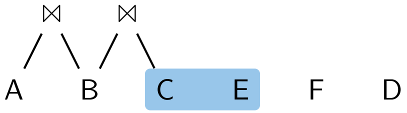
- Optimally combine optimal solutions for subchains



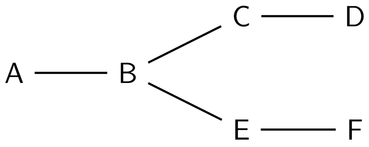


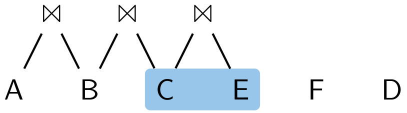
- Optimally combine optimal solutions for subchains



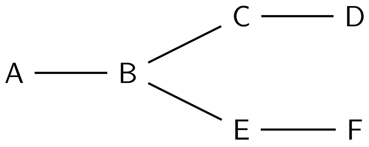


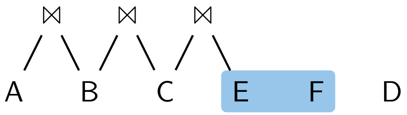
- Optimally combine optimal solutions for subchains



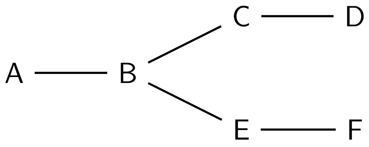


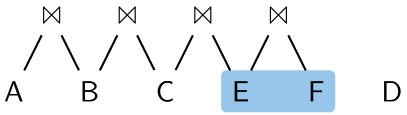
- Optimally combine optimal solutions for subchains



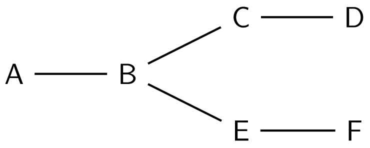


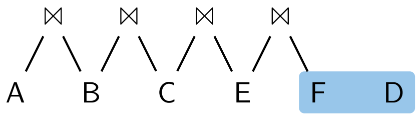
- Optimally combine optimal solutions for subchains



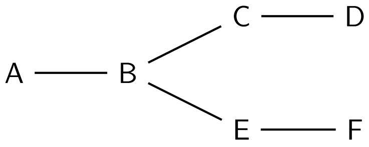


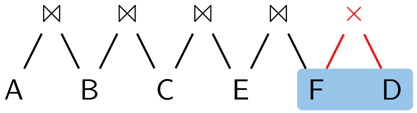
- Optimally combine optimal solutions for subchains



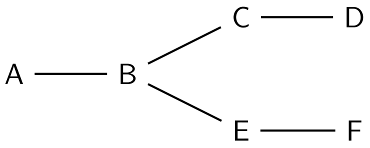


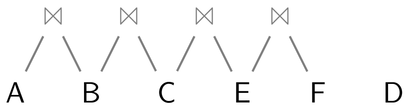
- Optimally combine optimal solutions for subchains



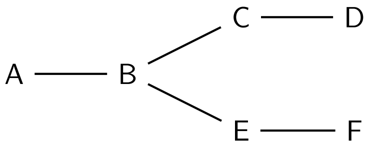


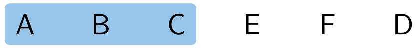
- Optimally combine optimal solutions for subchains



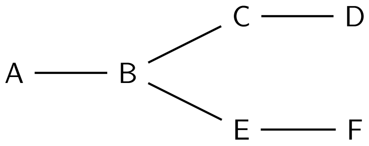


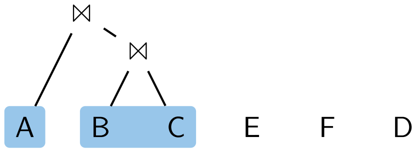
- Optimally combine optimal solutions for subchains



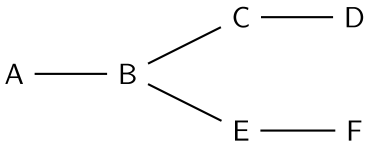


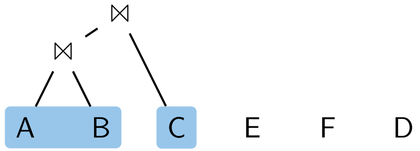
- Optimally combine optimal solutions for subchains



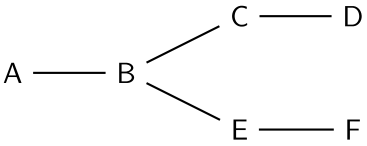


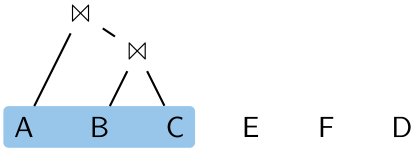
- Optimally combine optimal solutions for subchains



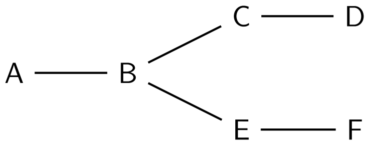


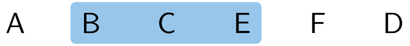
- Optimally combine optimal solutions for subchains



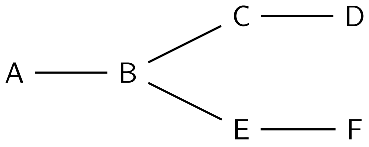


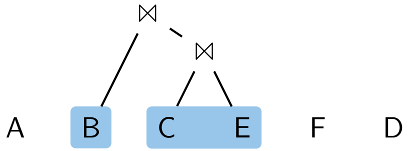
- Optimally combine optimal solutions for subchains



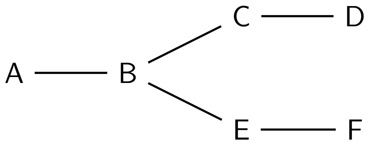


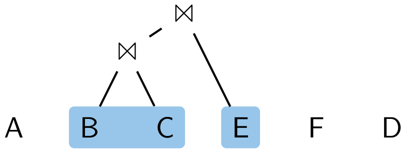
- Optimally combine optimal solutions for subchains



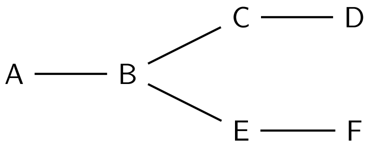


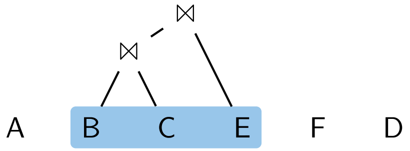
- Optimally combine optimal solutions for subchains



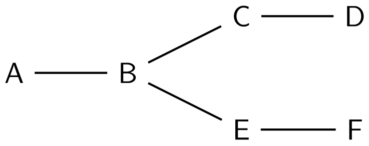


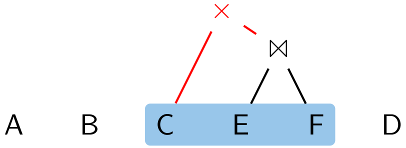
- Optimally combine optimal solutions for subchains



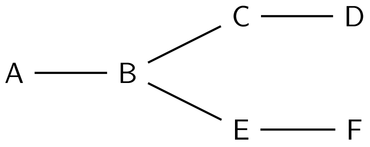


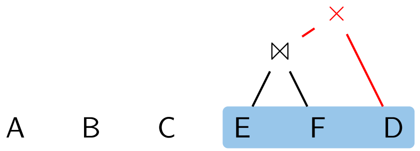
- Optimally combine optimal solutions for subchains



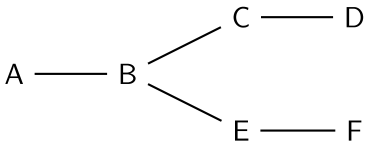


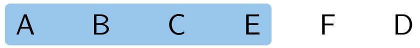
- Optimally combine optimal solutions for subchains



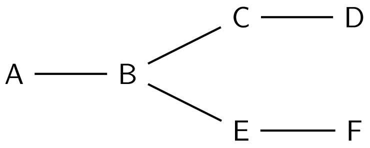


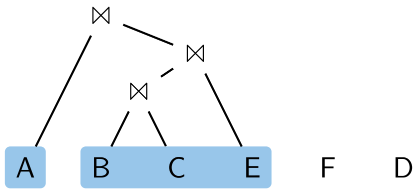
- Optimally combine optimal solutions for subchains



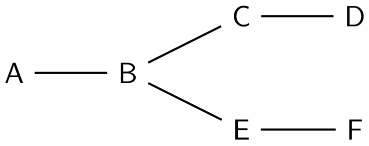


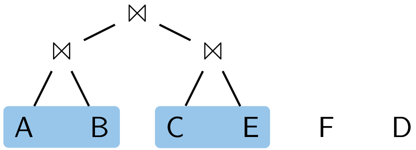
- Optimally combine optimal solutions for subchains



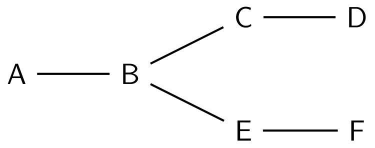


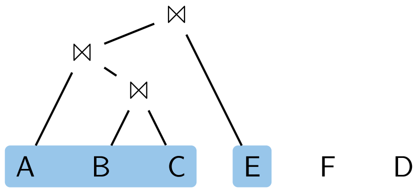
- Optimally combine optimal solutions for subchains



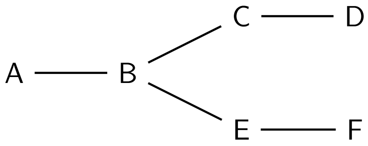


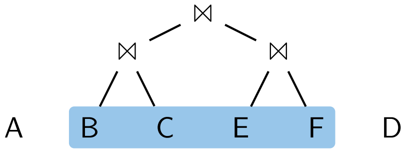
- Optimally combine optimal solutions for subchains



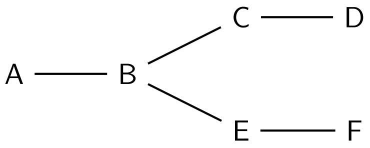


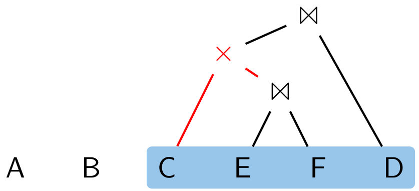
- Optimally combine optimal solutions for subchains



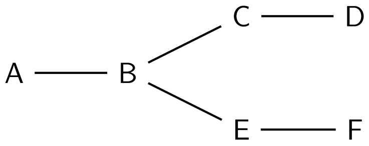


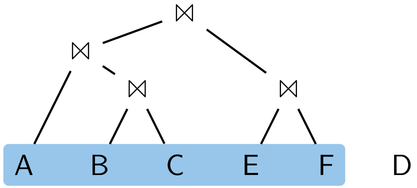
- Optimally combine optimal solutions for subchains



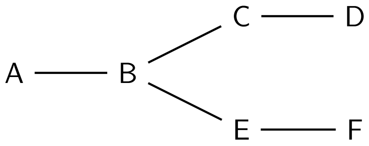


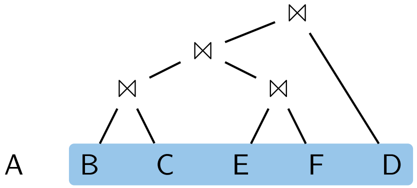
- Optimally combine optimal solutions for subchains



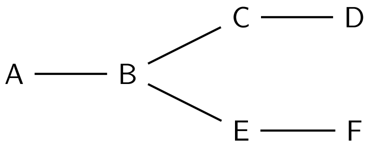


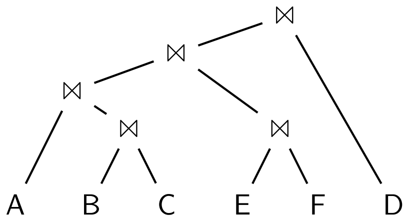
- Optimally combine optimal solutions for subchains





- Optimally combine optimal solutions for subchains





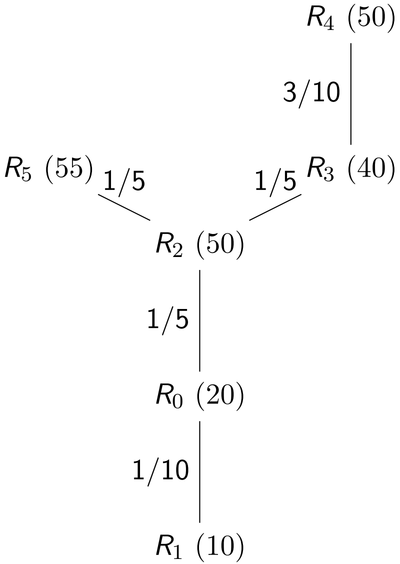
Query Optimization: Exercise

Session 9

Maximilian Rieger

December 22, 2023

Exercise [lindp]



- linearization rooted in R_4:
- normalize $R_0 R_1$, rank: $3/8$
- linearization: $R_4 R_3 R_2 R_0 R_1 R_5$
- cost:

```
  |4   3    2     0     1      5
  |0   1    2     3     4      5
--+-----------------------------
0 |0 600 6400 24800 24820 267020
1 |    0  400  1800  1820  19120
2 |         0   200   220   2420
3 |               0    20   1120
4 |                     0    550
5 |                            0
```

- Tree: $(R_4 \bowtie R_3) \bowtie (R_2(R_0 \bowtie R_1) \bowtie R_5)$

Cardinalities

```
  | 4   3    2     0     1      5
  | 0   1    2     3     4      5
--+----------------------------
0 |50 600 6000 24000 24000 264000
1 |    40  400  1600  1600  17600
2 |         50   200   200   2200
3 |               20    20   1120
4 |                     10    550
5 |                            55
```

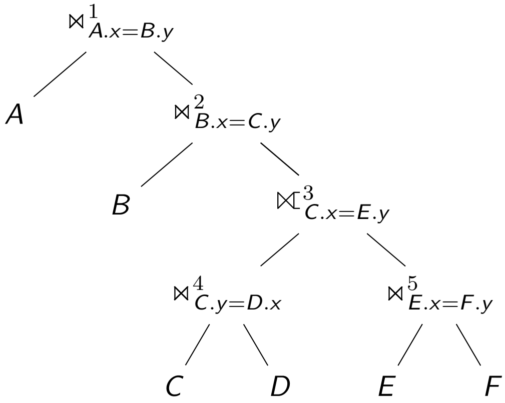
Best subsequences

```
  |4 3 2 0 1 5
  |0 1 2 3 4 5
--+-----------
0 |  0 0 1 1 1
1 |    1 1 1 2
2 |      2 2 4
3 |        3 4
4 |          4
5 |
```

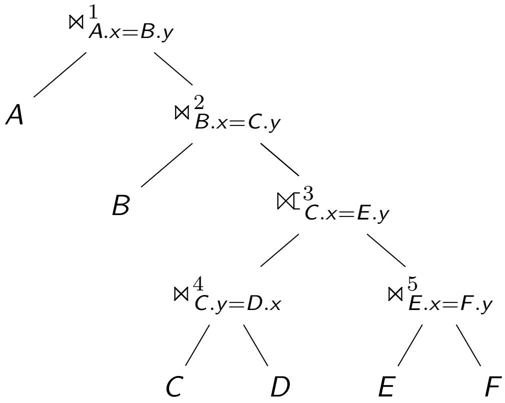
Cost

```
  |4   3    2     0     1      5
  |0   1    2     3     4      5
--+----------------------------
0 |0 600 6400 24800 24820 267020
1 |    0  400  1800  1820  19120
2 |         0   200   220   2420
3 |               0    20   1120
4 |                     0    550
5 |                            0
```

Exercise [restrictions]



- Syntactic eligibility set - relations that have to be in the input for the predicates to be syntactically valid
- Total eligibility set - additionally captures reordering restrictions, construct bottom-up
- Conflicts: $⟕^3_{C.x=E.y}$ and $\bowtie^4_{C.y=D.x}$, $⟕^3_{C.x=E.y}$ and $\bowtie^2_{B.x=C.y}$, $⟕^3_{C.x=E.y}$ and $\bowtie^1_{A.x=B.y}$



	SES	TES
$\bowtie^1_{A.x=B.y}$	$\{A, B\}$	$\{A, B, C, D, E\}$
$\bowtie^2_{B.x=C.y}$	$\{B, C\}$	$\{B, C, D, E\}$
$⟖^3_{C.x=E.y}$	$\{C, E\}$	$\{C, D, E\}$
$\bowtie^4_{C.y=D.x}$	$\{C, D\}$	$\{C, D\}$
$\bowtie^5_{E.x=F.y}$	$\{E, F\}$	$\{E, F\}$

- Finding conflicts is tricky [2]

Transformative Approaches

Explore the search space by directly applying equivalences to the initial join tree.

- start with the original tree
- apply equivalences (e.g. commutativity, associativity) to find a cheaper plan
- easy to integrate new equivalences
- hard to avoid generating trees multiple times — $\mathcal{O}(4^n)$ duplicates vs. $\mathcal{O}(3^n)$ entries in memo structure
- avoiding duplicates (by selectively disabling transformations) makes introduction of new equivalences hard

Memoizing Algorithm

```
ExhaustiveTransformation2(Query Graph $G$)
**Input:** a query specification for relations $\{R_1, \ldots, R_n\}$.
**Output:** an optimal join tree
initialize MEMO structure
ExploreClass($\{R_1, \ldots, R_n\}$)
**return** $\arg\min_{T \in \text{class } \{R_1, \ldots, R_n\}} C(T)$
```

- stored an arbitrary join tree in the memo structure
- explores alternatives recursively

Memoizing Algorithm (2)

```
ExploreClass(C)
Input: a class $\mathcal{C} \subseteq \{R_1, \ldots, R_n\}$
Output: none, but has side-effect on MEMO-structure
while not all join trees in C have been explored {
    choose an unexplored join tree $T$ in $\mathcal{C}$
    ApplyTransformation2($T$)
    mark $T$ as explored
}
```

- considers all alternatives within one class
- transformations themselves are done in ApplyTransformation2

Memoizing Algorithm (3)

```
ApplyTransformations2($T$)
Input: a join tree of a class $\mathcal{C}$
Output: none, but has side-effect on MEMO-structure
ExploreClass(left-child($T$))
ExploreClass(right-child($T$));
for each transformation $\mathcal{T}$ and class member of child classes {
    for each $T'$ resulting from applying $\mathcal{T}$ to $T$ {
        if $T'$ not in MEMO structure {
            add $T'$ to class $\mathcal{C}$ of MEMO structure
        }
    }
}
```

Rule Set RS-1

T_1: Commutativity $C_1 \bowtie_0 C_2 \rightsquigarrow C_2 \bowtie_1 C_1$

Disable all transformations T_1, T_2, and T_3 for $\bowtie_1$.

T_2: Right Associativity $(C_1 \bowtie_0 C_2) \bowtie_1 C_3 \rightsquigarrow C_1 \bowtie_2 (C_2 \bowtie_3 C_3)$

Disable transformations T_2 and T_3 for $\bowtie_2$ and enable all rules for $\bowtie_3$.

T_3: Left associativity $C_1 \bowtie_0 (C_2 \bowtie_1 C_3) \rightsquigarrow (C_1 \bowtie_2 C_2) \bowtie_3 C_3$

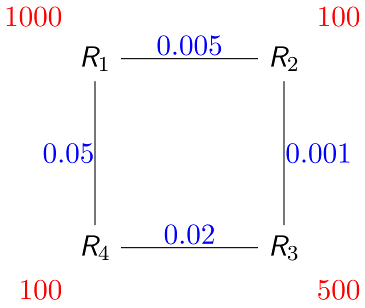
Disable transformations T_2 and T_3 for $\bowtie_3$ and enable all rules for $\bowtie_2$.

Example for chain $R_1 - R_2 - R_3 - R_4$

Class	Initialization	Transformation	Step
$\{R_1, R_2, R_3, R_4\}$	$\{R_1, R_2\} \bowtie_{111} \{R_3, R_4\}$	$\{R_3, R_4\} \bowtie_{000} \{R_1, R_2\}$	3
		$R_1 \bowtie_{100} \{R_2, R_3, R_4\}$	4
		$\{R_1, R_2, R_3\} \bowtie_{100} R_4$	5
		$\{R_2, R_3, R_4\} \bowtie_{000} R_1$	8
		$R_4 \bowtie_{000} \{R_1, R_2, R_3\}$	10
$\{R_2, R_3, R_4\}$		$R_2 \bowtie_{111} \{R_3, R_4\}$	4
		$\{R_3, R_4\} \bowtie_{000} R_2$	6
		$\{R_2, R_3\} \bowtie_{100} R_4$	6
		$R_4 \bowtie_{000} \{R_2, R_3\}$	7
$\{R_1, R_3, R_4\}$			
$\{R_1, R_2, R_4\}$			
$\{R_1, R_2, R_3\}$		$\{R_1, R_2\} \bowtie_{111} R_3$	5
		$R_3 \bowtie_{000} \{R_1, R_2\}$	9
		$R_1 \bowtie_{100} \{R_2, R_3\}$	9
		$\{R_2, R_3\} \bowtie_{000} R_1$	9
$\{R_3, R_4\}$	$R_3 \bowtie_{111} R_4$	$R_4 \bowtie_{000} R_3$	2
$\{R_2, R_4\}$			
$\{R_2, R_3\}$			
$\{R_1, R_4\}$			
$\{R_1, R_3\}$			
$\{R_1, R_2\}$	$R_1 \bowtie_{111} R_2$	$R_2 \bowtie_{000} R_1$	1

Generating Permutations

- Anytime algorithm to find optimal left deep tree
- Keep current prefix and the rest of relations
- Extend the prefix only if exchanging the last two relations does not result in a cheaper sequence



R_1
R_1, R_2
R_1, R_2, R_3
$R_1, R_2, R_3, R_4 \rightarrow 775$
R_1, R_2, R_4
$R_1, R_2, R_4, R_3 \rightarrow 3025$ X
R_1, R_3
R_1, R_3, R_2 ↯
R_1, R_3, R_4 ↯
R_1, R_4
R_1, R_4, R_2 ↯
R_1, R_4, R_3
$R_1, R_4, R_3, R_2 \rightarrow 55025$ X
...

Random Plans

- generating random trees with cross-products is easy
- no algorithm known to uniformly generate random trees without cross-products for all possible shapes of querygraphs
- complicated algorithm published to uniformly generate such trees for acyclic querygraphs [1]

Generating random trees with cross-products:

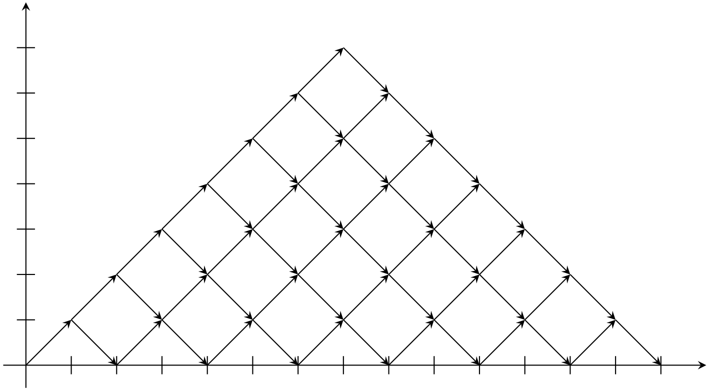
- Generate a tree, then generate a permutation: $C(n-1)$ trees, $n!$ permutations
- Pick a random number $b \in [0, C(n-1)[$, *unrank* b into tree
- Pick a random number $p \in [0, n![$, *unrank* p into permutation
- Attach the permutation to the leaves of the tree

Unranking Permutations

```
Unrank($n, r$)
Input:   the number $n$ of elements to be permuted
         and the rank $r$ of the permutation to be constructed
Output: a permutation $\pi$
for each $0 \leq i < n$
   $\pi[i] = i$
for each $n \geq i > 0$ descending {
   swap($\pi[i-1]$, $\pi[r \mod i]$)
   $r = \lfloor r/i \rfloor$
}
return $\pi$;
```

Unranking Treeshapes

- every tree is a word in $\{(,)\}$, closing parentheses must be matched by an already opened
- map such words to the grid, every step up is $($, down $)$
- the number of different paths q can be computed with formulas (see lectures) or right-to-left by hand
- Procedure: start at (0,0), walk up as long as rank is smaller than q. When it is greater or equal, step down, *rank*=*rank*-q



Quick-Pick

- Quickly generate pseudo-random plans
- Pick joins at random
- Fast, but not uniformly distributed
- Very similar to GOO

Next Weeks Exercise

- unrank a permutation
- unrank a tree
- apply transformations with rule set 1

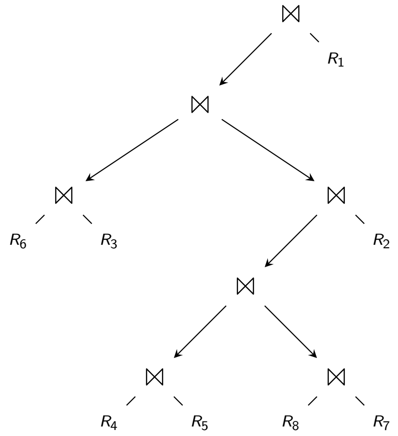
Query Optimization: Exercise

Session 10

Altan Birler

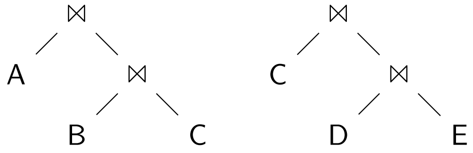
January 12, 2024

Anchored Lists



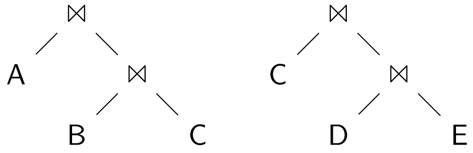
- How does the anchored list rooted in R_7 look like?
  $(< R_1, R_6 \bowtie R_3, R_2, R_4 \bowtie R_5, R_8 >, R_7)$
- How does the anchored list rooted in R_1 look like?
  $(< (R_6 \bowtie R_3) \bowtie (((R_4 \bowtie R_5) \bowtie (R_8 \bowtie R_7)) \bowtie R_2) >, R_1)$

List Merge



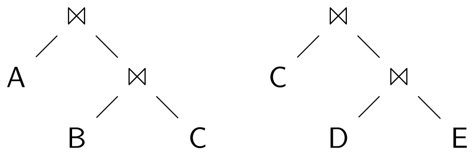
- Write the trees in anchored list notation and merge the subtrees using $\alpha = [1, 1]$

List Merge



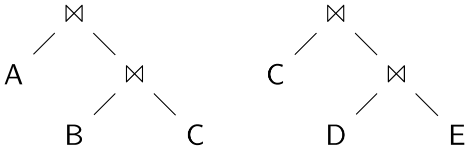
- Write the trees in anchored list notation and merge the subtrees using $\alpha = [1, 1]$
- $(< A, B >, C)$

List Merge



- Write the trees in anchored list notation and merge the subtrees using $\alpha = [1, 1]$
- $(< A, B >, C)$
- $(< (D \bowtie E) >, C)$

List Merge



- Write the trees in anchored list notation and merge the subtrees using $\alpha = [1, 1]$
- $(< A, B >, C)$
- $(< (D \bowtie E) >, C)$
- $(< A, (D \bowtie E), B >, C)$

so far:

- GreedyJoinOrdering-{1,2,3} heuristics
- Greedy Operator Ordering (GOO) [?]
- IKKBZ [?] [?]
- Maximum Value Precedence (MVP) [?]
- Dynamic Programming [?, ?, ?, ?, ?]
- Adaptive Optimization [?, ?]
- Generating Permutations
- Transformative Approaches [?]
- Random Plans

today:

- Metaheuristics [?]
- Iterative DP [?]
- Order Preserving Joins

Metaheuristics

- Iterative Improvement: Go to random neighbor only if it is better
- Simulated Annealing: Go to random neighbor with probability 1 if it is better or with a certain decreasing probability (reducing temperature) if it is worse
- Tabu Search: Go to best neighbor

Genetic Algorithms

Big picture

- Create a "population", i.e. create p random join trees
- Encode them using ordered list or ordinal number encoding
- Create the next generation
  - Randomly mutate some members (e.g. exchange two relations)
  - Pairs members of the population and create "crossovers"
- Select the best, kill the rest (chance of survival depending on cost)

Details

- Encoding
- Crossover

Encoding

Ordered lists

- Simple
- Left-deep trees: Straight-forward
- Bushy trees: Label edges in join-graph, encode the processing tree just like the execution engine will evaluate it

Ordinal numbers

- Are slightly more complex
- Manipulate a list of relations (careful: indexes are 1-based)
- Left-deep trees: $(((R_1 \bowtie R_4) \bowtie R_3) \bowtie R_2) \bowtie R_5 \;\mapsto\; 13211$
- Bushy trees: $(R_2 \bowtie (R_1 \bowtie R_3)) \bowtie (R_4 \bowtie R_5) \;\mapsto\; 13\,21\,23\,12$

Crossover

Subsequence exchange for ordered list encoding

- Select subsequence in parent 1, e.g. $abc\underline{def}gh$
- Reorder subsequence according to the order in parent 2

Subsequence exchange for ordinal number encoding

- Swap two sequences of same length and same offset
- What if we get duplicates?

Subset exchange for ordered list encoding

- Find random subseqeces in both parents that have the same length and contain the same relations
- Exchange them to create two children

Combinations

- 2PO (II and then SA)
- AB Algorithm (IKKBZ and then II)
- Toured SA (SA for each join sequence produced by GreedyJoinOrdering-3)
- GOO-II (run II on the result of GOO)

Iterative Dynamic Programming

- IDP-1
  - build solutions up to size k using DP
  - replace the cheapest with a compound relation
  - repeat until all relations are covered
- IDP-2
  - greedily build a solution for the complete query (e.g. using GOO)
  - find the most expensive subtree that covers at most k relations
  - optimize that subtree using DP
  - replace the original subtree a compound relation representing the DP solution
  - repeat until a single compound relation remains (or out of budget)

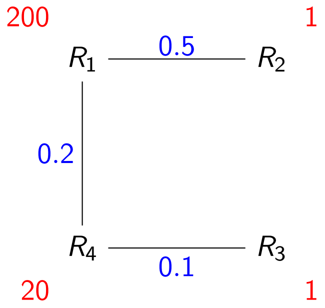
Order-Preserving Joins

Same algorithm that we used for LinDP! Use cases:

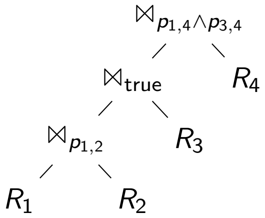
1. Chain queries (can be extended to cycles)
2. Order preserving joins
3. LinDP (where we have to preserve the linearized order)

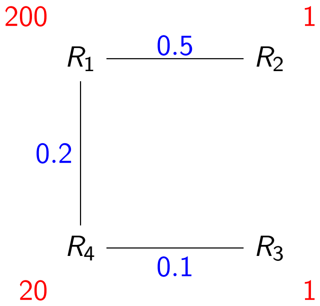
An example: Consider the following *sequence* of relations R_1, R_2, R_3, R_4 with cardinalities $|R_1| = 200$, $|R_2| = 1$, $|R_3| = 1$, $|R_4| = 20$ and join selectivities $f_{1,2} = 0.5$, $f_{1,4} = 0.2$, $f_{3,4} = 0.1$.

Give the fully-parenthesized, optimal join-expression that abides by this order. Use C_{out} as a cost function.



Let's start off with a cost analysis of the left-deep tree:





$C_{out} = 100 + 100 + 40 = 240$

```
01 for each $2 \le l \le 4$ ascending (in text: $2 \le l \le n$ )
02    for each $1 \le i \le 5 - l$ (in text: $1 \le i \le n - l + 1$)
03       $j = i + l - 1$
06       $s[i,j]$=cardinality derived from $s[i, j-1]$ and $s[j,j]$ and additional predicates
07       $c[i,j]=\infty$
08       for each $i \le k < j$
09          $q = c[i,k] + c[k+1,j]$+costs for $s[i,j]$
10          if $q < c[i,j]$
11             c[i,j]=q
12             t[i,j]=k
```

statistics s

200	100	100	40
	1	1	2
		1	2
			20

costs c

0	100	101	43
	0	1	3
		0	2
			0

split points t

	1	1	1
		2	3
			3

```
for each $2 \leq l \leq 4$ ascending (in text: $2 \leq l \leq n$ )
   for each $1 \leq i \leq 5-l$ (in text: $1 \leq i \leq n-l+1$)
      $j = i + l - 1$
      $s[i,j]$=cardinality derived from $s[i,j-1]$ and $s[j,j]$ and additional predicates
      $c[i,j]$=$\infty$
      for each $i \leq k < j$
         $q = c[i,k] + c[k+1,j]$+costs for $s[i,j]$
         if $q < c[i,j]$
            c[i,j]=q
            t[i,j]=k
```

statistics s

200	100	100	40
	1	1	2
		1	2
			20

costs c

0	100	101	43
	0	1	3
		0	2
			0

trees t

R_1	$R_1 \bowtie R_2$	$R_1 \bowtie (R_2 \bowtie R_3)$	$R_1 \bowtie ((R_2 \bowtie R_3) \bowtie R_4)$
	R_2	$R_2 \bowtie R_3$	$(R_2 \bowtie R_3) \bowtie R_4$
		R_3	$R_3 \bowtie R_4$
			R_4

The values of t are:

i/j	1	2	3	4
1		1	1	1
2			2	3
3				3
4				

```
ExtractPlan($R = \{R_1, \ldots, R_n\}$,$t$)
Input:   a set of relations, array $t$
Output: a bushy join tree
return ExtractPlanRec($R$,$t$,1,$n$)

ExtractPlanRec($R = \{R_1, \ldots, R_n\}$,$t$,$i$,$j$)
if $i < j$
    $T_1$ =ExtractPlanRec($R$,$t$,$i$,$t[i,j]$)
    $T_2$ =ExtractPlanRec($R$,$t$,$t[i,j]+1,j$)
    return $T_1 \bowtie^L_{p[i,j]} T_2$
else
    return $\sigma_{p[i,j]} R_i$
```

The values of t are:

i/j	1	2	3	4
1		1	1	1
2			2	3
3				3
4				

```
extract-subplan(..., i=1, j=4)
    extract-subplan(..., i=1, j=1)
    extract-subplan(..., i=2, j=4)
        extract-subplan(..., i=2, j=3)
            extract-subplan(..., i=2, j=2)
            extract-subplan(..., i=3, j=3)
        **return** $(R_2 \bowtie_{\text{true}} R_3)$
        extract-subplan(..., i=4, j=4)
    **return** $((R_2 \bowtie_{\text{true}} R_3) \bowtie_{p_{3,4}} R_4)$
**return** $(R_1 \bowtie_{p_{1,2} \wedge p_{1,4}} ((R_2 \bowtie_{\text{true}} R_3) \bowtie_{p_{3,4}} R_4))$
```

The total cost of this plan is $c[1,4] = 43$.

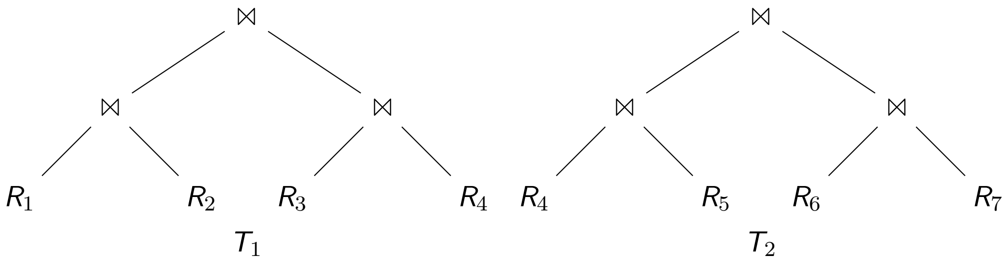
Query Optimization: Exercise

Session 11

Maximilian Rieger

January 17, 2024

Exercise [uniformrandom]



As anchored list:

$$T_1 :(< (R_1 \bowtie R_2), R_3 >, R_4)$$

$$T_2 :(< (R_6 \bowtie R_7), R_5 >, R_4)$$

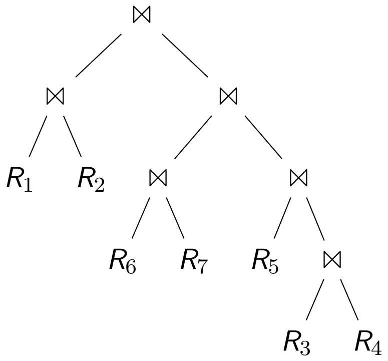
$$T_1 :(< (R_1 \bowtie R_2), R_3 >, R_4)$$
$$T_2 :(< (R_6 \bowtie R_7), R_5 >, R_4)$$

Merge using $\alpha = [1, 0, 1]$:

$$< (R_1 \bowtie R_2), (R_6 \bowtie R_7), R_5, R_3 >$$

As anchored lists: $(< (R_1 \bowtie R_2), (R_6 \bowtie R_7), R_5, R_3 >, R_4)$

Anchored list:
$(< (R_1 \bowtie R_2), (R_6 \bowtie R_7), R_5, R_3 >, R_4)$
The join tree:



Exercise [geneticencoding]

For the bushy tree $(R_4 \bowtie_{p_1} (R_3 \bowtie_{p_2} R_5)) \bowtie_{p_3} (R_1 \bowtie_{p_4} R_2)$

- Ordered List Encoding: 2 1 4 3
- Ordinal Number Encoding: 35 41 23 12
  - $R_1,\ R_2,\ R_3,\ R_4,\ R_5$
  - $R_3 \bowtie R_5,\ R_1,\ R_2,\ R_4$
  - $R_4 \bowtie (R_3 \bowtie R_5),\ R_1,\ R_2$
  - $R_4 \bowtie (R_3 \bowtie R_5),\ R_1 \bowtie R_2$
  - $(R_4 \bowtie (R_3 \bowtie R_5)) \bowtie (R_1 \bowtie R_2)$

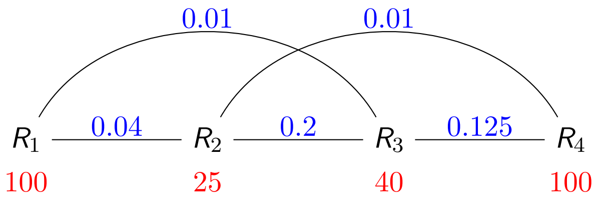
For the left-deep tree $(((R_4 \bowtie R_1) \bowtie R_3) \bowtie R_2)$

- Ordered List Encoding: 4 1 3 2
- Ordinal Number Encoding: 4 1 2 1
    - R_1, R_2, R_3, R_4
    - R_1, R_2, R_3
    - R_2, R_3
    - R_2

Give the bushy join tree for the ordinal number encoding “35 13 23 12”

- $R_1,\ R_2,\ R_3,\ R_4,\ R_5$
- $R_3 \bowtie R_5,\ R_1,\ R_2,\ R_4$
- $(R_3 \bowtie R_5) \bowtie R_2,\ R_1,\ R_4$
- $(R_3 \bowtie R_5) \bowtie R_2,\ R_1 \bowtie R_4$
- $((R_3 \bowtie R_5) \bowtie R_2) \bowtie (R_1 \bowtie R_4)$

Exercise [orderpreserving]



- $[1,4]$ Split 3: $\{R_1, R_2, R_3\} \bowtie R_4$
- $[1,3]$ Split 2: $\{R_1, R_2\} \bowtie R_3$
- $[1,2]$ Split 1: $R_1 \bowtie R_2$
- $((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4$
- Cost: 109

cardinalities s

	1	2	3	4
1	100	100	8	1
2		25	200	25
3			40	500
4				100

cost c

	1	2	3	4
1	0	100	108	109
2		0	200	225
3			0	500
4				0

split points t

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

DPweb

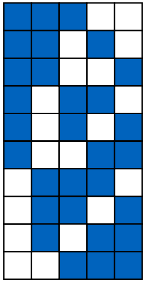
Accessing the Data

Direct, Uniform, Distinct: Yao

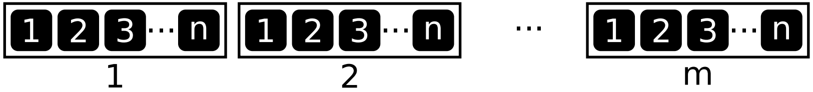
Given a set of n elements, how many distinct k-element subsets can be formed?

$$\binom{n}{k} = \frac{n!}{(n-k)! \cdot k!}$$

Example: Choose 3 out of 5: $\binom{5}{3} = \frac{5!}{2! \cdot 3!} = \frac{120}{2 \cdot 6} = 10$



Given m pages with n tuples on each page, e.g. a total of $N = m \cdot n$ tuples:



- How many distinct subsets of size k exist? $\binom{N}{k}$
- How many distinct subsets of size k exist, where a page does not contain any of the chosen tuples? Choose k from all but one page, i.e. from $N - n$ tuples: $\binom{N-n}{k}$
  So the probability that a page contains none of the k tuples is

$$p := \frac{\binom{N-n}{k}}{\binom{N}{k}}$$

- What is the probability that a certain page contains at least one tuple? $1 - p$...unless all pages have to be involved $(k > N - n)$.
- Multiplied by the number of pages, we get the number of qualifying pages, denoted $\overline{\mathcal{Y}}_n^{N,m}(k) = m \cdot \mathcal{Y}_n^N(k)$.

Yao overview:

N Tuples in relation, m pages, n tuples per page

no tuple from page selected : $p = \frac{\binom{N-n}{k}}{\binom{N}{k}} = \prod_{i=0}^{k-1} \frac{N-n-i}{N-i}$

page contains ≥ 1 tuple : $\mathcal{Y}_n^N(k) = \begin{cases} 1-p, & k \leq N-n \\ 1, & k > N-n \end{cases}$

number of pages : $\overline{\mathcal{Y}}_n^{N,m}(k) = m \cdot \mathcal{Y}_n^N(k)$

Waters approximation : $p \approx (1 - \frac{k}{N})^n$

Bernstein approximation : $\overline{\mathcal{Y}}_n^{N,m}(k) \approx \begin{cases} k, & k < \frac{m}{2} \\ \frac{k+m}{3}, & \frac{m}{2} \leq k < 2m \\ m, & 2m \leq k \end{cases}$

Let $m = 50,\ n = 1000 \Rightarrow N = 50k,\ k = 100$

Yao (exact) : $p = \frac{\binom{N-n}{k}}{\binom{N}{k}} = \prod_{i=0}^{k-1} \frac{N-n-i}{N-i} = \prod_{i=0}^{99} \frac{49k-i}{50k-i} = 13.2\%$

Waters : $p \approx (1 - \frac{k}{N})^n \approx 13.5\%$

Query Optimization: Exercise Session 12

Altan Birler

January 26, 2024

1 Linearity of expectation

Let X_1, X_2 be a random variables (possibly dependent). Then,

$$\mathbb{E}[X_1 + X_2] = \mathbb{E}[X_1] + \mathbb{E}[X_2] \tag{1}$$

2 Uniform, distinct

There are m buckets with n items each with $N = mn$ total items. k distinct items are randomly picked from N with equal probability. What is the expected number of buckets from which items have been picked?

Define

$$X_i := \begin{cases} 1 & \text{if bucket } i \text{ has been picked from} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Then, the number of qualifying buckets $\overline{\mathcal{Y}}_n^{N,m}(k)$ is given by

$$\overline{\mathcal{Y}}_n^{N,m}(k) = \mathbb{E}\left[\sum_{i=1}^{m} X_i\right] = \sum_{i=1}^{m} \mathbb{E}[X_i] \tag{3}$$

Since the probabilities are uniform, i.e. buckets are symmetric, we can simplify by picking an arbitary bucket

$$\overline{\mathcal{Y}}_n^{N,m}(k) = \sum_{i=1}^{m} \mathbb{E}[X_i] = m\mathbb{E}[X_1] \tag{4}$$

What remains is to compute $\mathbb{E}[X_1]$ (which corresponds to $\mathcal{Y}_n^N(k)$ from the lecture).

$$\mathbb{E}[X_1] = \mathbb{P}[X_1 = 1] \tag{5}$$

Computing $\mathbb{P}[X_1 = 1]$, the probability that bucket 1 qualifies, is tricky in general. If $k > N - n$, then all buckets must have been picked from, including bucket 1. Otherwise, the bucket qualifies if 1, 2, or any number of items have been picked from it. Computing the complement is simplier, as we just have to find the number of events wherein no items have been picked from the bucket. Assuming $k \leq N - n$

$$\mathbb{P}[X_1 = 1] = 1 - \mathbb{P}[X_1 = 0] \tag{6}$$

$$\mathbb{P}[X_1 = 0] = \frac{\text{\# pickings where no items are from bucket 1}}{\text{total \# of pickings}} \tag{7}$$

Total number of pickings is $\binom{N}{k}$. Pickings without items from bucket 1 is $\binom{N-n}{k}$, where n is the number of items in bucket 1 (as in any bucket). Putting it all together, we derive Yao's formula:

$$\overline{\mathcal{Y}}_n^{N,m}(k) = m\left(1 - \frac{\binom{N-n}{k}}{\binom{N}{k}}\right) \tag{8}$$

Waters provides an approximation for $\mathbb{P}[X_1 = 0]$. First, we define random variables for picking individual items from bucket 1:

$$X_{1,j} \coloneqq \begin{cases} 1 & \text{if item } j \text{ from bucket 1 is picked} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

$$\begin{aligned} \mathbb{P}[X_1 = 0] &= \mathbb{P}[X_{1,1} = 0 \wedge X_{1,2} = 0 \wedge \ldots \wedge X_{1,n} = 0] && (10) \\ &\approx \mathbb{P}[X_{1,1} = 0] \cdot \mathbb{P}[X_{1,2} = 0] \cdot \ldots \cdot \mathbb{P}[X_{1,n} = 0] && (11) \\ &= \left(1 - \frac{k}{N}\right)^n && (12) \end{aligned}$$

Since the random variables of $X_{1,j}$ are not independent, the result is an approximation. Bernstein provides a piecewise linear approximation.

3 Uniform, non-distinct

The same as above, but now the same items can be picked multiple times. One way to think about this is that every item is assigned a count of how many times it has been picked, and the sum of these counts is k. The expected number of buckets is $\overline{Cheung}_n^{N,m}(k)$.

$$X_i \coloneqq \begin{cases} 1 & \text{if bucket } i \text{ has been picked from} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

$$\overline{Cheung}_n^{N,m}(k) = \sum_{i=1}^{m} \mathbb{E}[X_i] = m\mathbb{E}[X_1] \tag{14}$$

$$\overline{Cheung}_n^{N,m}(k) = m\,(1 - \mathbb{P}[X_1 = 0]) \tag{15}$$

$$= m\left(1 - \frac{\#\text{ pickings without bucket 1}}{\text{total }\#\text{ of pickings}}\right) \tag{16}$$

What remains is to compute number of pickings with duplicates. This is equivalent to the stars and bars problem, where we have k stars and $N-1$ bars.

$$\overline{Cheung}_n^{N,m}(k) = m\left(1 - \frac{\frac{(N-n-1+k)!}{(N-n-1)!k!}}{\frac{(N-1+k)!}{(N-1)!k!}}\right) \tag{17}$$

$$= m\left(1 - \frac{\binom{N-n-1+k}{k}}{\binom{N-1+k}{k}}\right) \tag{18}$$

Cardenas provides and approximation:

$$X_{1,j} \coloneqq \begin{cases} 1 & \text{if the jth pick is in bucket i} \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

$$\mathbb{P}[X_1 = 0] \approx \mathbb{P}[X_{1,1} = 0 \land X_{1,2} = 0 \land \ldots \land X_{1,k} = 0] \tag{20}$$

$$\approx \mathbb{P}[X_{1,1} = 0] \cdot \mathbb{P}[X_{1,2} = 0] \cdot \ldots \cdot \mathbb{P}[X_{1,k} = 0] \tag{21}$$

$$= \left(1 - \frac{n}{N}\right)^k \tag{22}$$

Note how the Cardenas approximation allows for duplicates while Waters does not.

Picking a k-multiset from an N-set is equivalent to Cheung with $n = 1$ and $m = N$.

4 Non-uniform, distinct

If each bucket contains a different number of items, the big picture remains similar. However, since the probabilities for individual buckets are different in this case, we can't just pick an arbitrary bucket to represent all other buckets.

5 Random sampling

(In the lecture, you were shown the hypergeometric distribution $\mathcal{X}_{n_j}^N$ under the section “non-uniform, distinct”. In the exercise, we will try to understand this distribution through a common usecase, random sampling.)

We have a relation R with N tuples. We apply a predicate p on this relation that qualifies n tuples. The selectivity of p is thus $\sigma_p = \frac{n}{N}$. From R, we randomly pick k distinct tuples. We call these k tuples a simple random sample (all sets of tuples of size k are equally likely to be the sample).

What is the expected number of tuples in the sample that qualify p? Quite ntuitively, $k \cdot \sigma_p = k \cdot \frac{n}{N}$ is the correct answer.

What is the distribution of the number of tuples in the sample that qualify p?

$$\mathbb{P}[X = x] = \frac{\#\text{ of samples with } x \text{ qualifying tuples}}{\text{total }\#\text{ of samples}} \tag{23}$$
$$= \frac{(x \text{ qualifying from } n) \cdot (k - x \text{ non-qualifying from } N - n)}{n \text{ from } N} \tag{24}$$
$$= \frac{\binom{n}{x}\binom{N-n}{k-x}}{\binom{N}{k}} \tag{25}$$

This is the hypergeometric distribution.

6 Vector of bits

We have a vector of B bits where only b random bits are set to 1. We want to analyze the number of zeroes between two consecutive 1s. (Or from the beginning to the first 1, or from the last 1 to the end. It does not matter. As an intuition, you can imagine that the sequence has a hidden 1s at the very beginning and the very end.)

What is the number of zeroes between two consecutive 1s in expectation? Since the result will be the same regardless of which pairs of 1s we pick, we just divide the total number of zeroes by the number of regions split up by 1s.

$$\mathbb{E}[\#\text{ of zeroes between two consecutive 1s}] = \frac{\#\text{ of zeroes}}{\#\text{ of regions}} \tag{26}$$
$$= \frac{B - b}{b + 1} \tag{27}$$
$$= \overline{\mathcal{B}}_b^B \tag{28}$$

Given an arbitary but fixed consecutive pair of 1s $\mathbb{1}_i$ and $\mathbb{1}_{i+1}$, what is the probability distribution of the number of zeroes between these two 1s?

$$\mathbb{P}[X = j] = \frac{\text{\# of bitvectors with } j \text{ zeroes between } \mathbb{1}_i \text{ and } \mathbb{1}_{i+1}}{\text{total \# of bitvectors}} \tag{29}$$

Total # of bitvectors is $\binom{B}{b}$. And there is a bijection between bitvectors V_1 with B bits, b ones and j zeroes between $\mathbb{1}_i$ and $\mathbb{1}_{i+1}$ and bitvectors V_2 with $B-b-1$ bits and $b-1$ ones. You can take any V_1 bitvector, remove $\mathbb{1}_i$ and the j zeroes between $\mathbb{1}_i$ and $\mathbb{1}_{i+1}$, and get a valid V_2 bitvector with $B-b-1$ bits and $b-1$ ones. Similarly, you can take any V_2 bitvector, insert a $\mathbb{1}$ and j zeroes before its ith one, and get a valid V_1 bitvector with B bits and b ones. Thus,

$$\mathbb{P}[X = j] = \frac{\binom{B-j-1}{b-1}}{\binom{B}{b}} \tag{30}$$

$$= \mathcal{B}_b^B(j) \tag{31}$$

Query Optimization: Exercise

Session 13

Maximilian Rieger

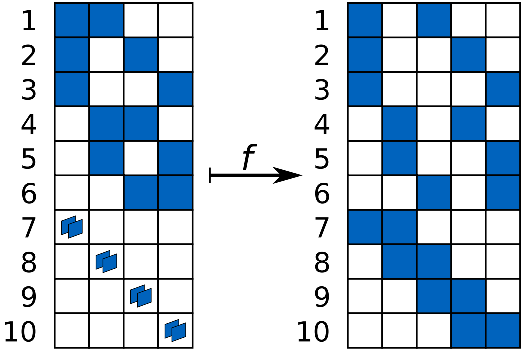
February 2, 2024

Accessing the Data

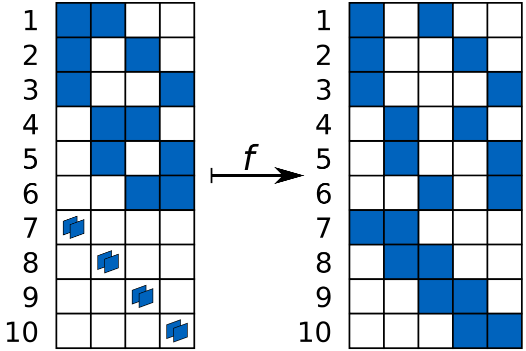
Direct, Uniform, Distinct: Yao

Direct, Uniform, Non-Distinct: Cheung

- Now *with replacement*: How many distinct *multisets* exist chosing k from n? As many as there are distinct sets chosing k from $n+k-1$!
- Bijection between multisets and sets. From multiset to set (x_i sorted ascending): $f : (x_1, x_2, \ldots, x_k) \mapsto (x_1 + 0, x_2 + 1, \ldots, x_k + (k-1))$
- Example: Choose 2 from 4
  - # sets: $\binom{4}{2}$
  - # multisets: $\binom{4+2-1}{2}$



- An alternative bijection is to represent the choice as a string of elements and bucket separators:



1. `x|x||`
2. `x||x|`
3. `x|||x`
4. `|x|x|`
5. `|x||x`
6. `||x|x`
7. `xx|||`
8. `|xx||`
9. `||xx|`
10. `|||xx`

- Like Yao, but not necessarily distinct
- Same formula as Yao, but:
  - No special case for $k > N - n$
  - We substitue N by $N + k - 1$ to compute $\tilde{p}$

$$p = \frac{\binom{N-n}{k}}{\binom{N}{k}} = \prod_{i=0}^{n-1} \frac{N-k-i}{N-i}$$
$$\tilde{p} = \frac{\binom{N-n+k-1}{k}}{\binom{N+k-1}{k}} = \prod_{i=0}^{n-1} \frac{N-1-i}{N-1+k-i}$$

- Cardenas approximation:

$$\tilde{p} \approx (1 - \frac{n}{N})^k$$

Exercise [cheung]

Given a relation with 3 pages and two tuples per page, compute the average number of accessed pages when reading 4 not necessarily distinct tuples.

- $N = 6,\ n = 2,\ k = 4,\ m = 3$

$$\tilde{p} = \frac{\binom{N-n+k-1}{k}}{\binom{N+k-1}{k}} = \frac{\binom{6-2+4-1}{4}}{\binom{6+4-1}{4}} = \frac{5}{18}$$

$$\begin{aligned}\overline{\text{Cheung}}_n^{N,m}(k) &= \overline{\text{Cheung}}_2^{6,3}(4) \\ &= 3\left(1-\tilde{p}\right) \\ &= \frac{13}{6}\end{aligned}$$

Exercise [bitvector]

Given a relation consecutively stored on 100 pages on disk, assuming 10 random pages to be read:

- Give the probability of j initial pages being skipped for $j \in \{0, 30, 60, 90\}$

$$\mathcal{B}_b^B(j) = \frac{\binom{B-j-1}{b-1}}{\binom{B}{b}}$$

  - 0.1 0.00327389 0.0000122421 0

Exercise [bitvector]

Given a relation consecutively stored on 100 pages on disk, assuming 10 random pages to be read:

- Estimate the number of *skipped* pages between two of the pages to be read

$$\overline{\mathcal{B}}_b^B = \sum_{j=0}^{B-b} j\mathcal{B}_b^B(j) = \frac{B-b}{b+1} = \frac{100-10}{10+1} \approx 8.18$$

- Estimate the number of pages between the beginning of the relation and the last page to be read, including the last page

$$B - \overline{\mathcal{B}}_b^B \approx 100 - 8.18 = 91.82$$

- Estimate the number of pages between the first and the last page to be read, including the first and the last page

$$B - 2\overline{\mathcal{B}}_b^B \approx 100 - 2 \cdot 8.18 = 100 - 16.36 = 83.64$$

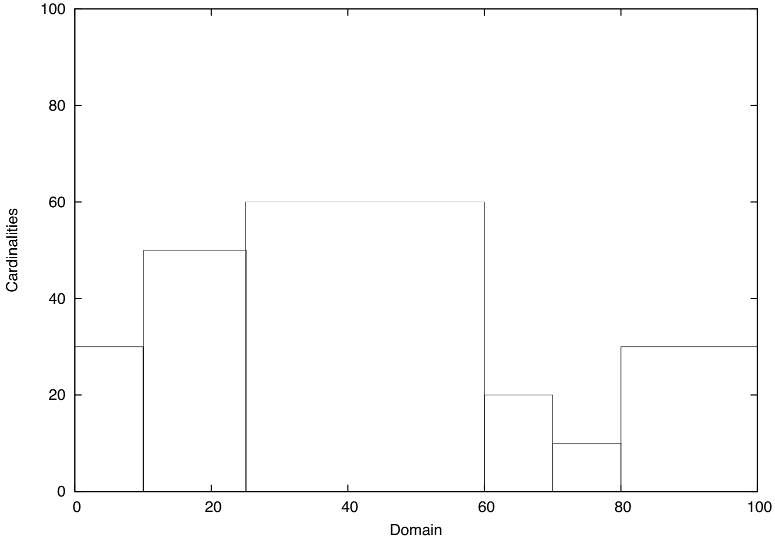
Histograms

A histogram $H_A : B \rightarrow \mathbb{N}$ over a relation R partitions the domain of the aggregated attribute A into disjoint buckets B, such that

$$H_A(b) = |\{r|r \in R \wedge R.A \in b\}|$$

and thus $\sum_{b \in B} H_A(b) = |R|$.

A rough histogram might look like this:



Given a histogram, we can approximate selectivities as follows:

$$A = c \qquad \frac{\sum_{b \in B : c \in b} H_A(b)}{\sum_{b \in B} H_A(b)}$$

$$A > c \qquad \frac{\sum_{b \in B : c \in b} \frac{\max(b) - c}{\max(b) - \min(b)} H_A(b) + \sum_{b \in B : \min(b) > c} H_A(b)}{\sum_{b \in B} H_A(b)}$$

$$A_1 = A_2 \qquad \frac{\sum_{b_1 \in B_1, b_2 \in B_2, b' = b_1 \cap b_2 : b' \neq \emptyset} \frac{\max(b') - \min(b')}{\max(b_1) - \min(b_1)} H_{A_1}(b_1) \frac{\max(b') - \min(b')}{\max(b_2) - \min(b_2)} H_{A_2}(b_2)}{\sum_{b_1 \in B_1} H_{A_1}(b_1) \sum_{b_2 \in B_2} H_{A_2}(b_2)}$$

HyperLogLog Sketches

- Problem: approximate the distinct count of values in a column
- Use little memory, support inserts
- Idea: Use a hash value and count the leading zeros. Store counts of observed leading zeros in bitset.
- The estimated number of distinct values is 2^{b_s} where b_s is the number of ones at the beginning of the bitset

HyperLogLog Sketches

- Idea: Use a hash value and count the leading zeros. Store the max count of observed leading zeros l.
- The estimated number of distinct values is 2^{b_s}
- Improve accuracy:
  - Use n bitmasks, randomly select one for each insert
  - The estimated number of distinct values is $n \cdot 2^{\frac{\sum_{i=1}^{n} b_s^{(i)}}{n}}$
  - error usually $< 10\%$ for $n = 64$
- Improve memory usage:
  - Store max number of leading zeros observed
  - As this is at most 64, we only need one byte per set

Cardinality estimation for complex predicates

- How to estimate `name like '%bob%' and x < 2`
- Sample:

Name	x
john	1
jane	2
alice	5
bob	3
mary	1
tom	2

- What if your sample is empty?
- Idea: Assume independence of attributes and evaluate each part of conjunction
- Estimated selectivity: $\frac{1}{6} \cdot \frac{2}{6}$
- Bad, but what can you do...

Unnesting

Query Optimization: Exercise

Session 14

Altan Birler

February 9, 2024

Repetition

Motivation

- declarative query has to be translated into an imperative, executable plan
- usually multiple semantically equivalent plans (search space)
- possibly huge differences in execution costs of different alternatives (asymptotic improvements)

Goal: find the cheapest of those plans

Query Graph

- undirected graph
- nodes: relations
- edges: predicates/joins
- different shapes (e.g. chain, star, tree, clique)
- shape influences size of the search space!
- Hypergraphs:
  - Complex join predicates
  - Reordering restrictions
  - Query simplification

Join Tree

- inner nodes: operators (e.g. join, cross product)
- leaves: relations
- different shapes
  - linear (left-deep, right-deep, zigzag)
  - bushy
- desired shape influences size of the search space
  - with cross products: number of tree shapes * number of leaf permutations
  - without cross products: depends on the shape of the query graph

Selectivity, Cardinality

$$f_p = \frac{|\sigma_p(R)|}{|R|}$$

$$f_{i,j} = \frac{|R_i \bowtie_{p_{i,j}} R_j|}{|R_i \times R_j|}$$

- Independence: Data distributions of different columns are independent.
- Uniformity: Values in a column are distributed uniformly.
- Containment: Smaller set is contained within the larger set.

Costs

$$C_{out}(R) = 0$$
$$C_{out}(R_i \bowtie R_j) = |R_i \bowtie R_j| + C_{out}(R_i) + C_{out}(R_j)$$

- more advanced cost functions for different physical join implementations
- properties
  - symmetry: $C(A \bowtie B) = C(B \bowtie A)$
  - ASI: rank function r such that $r(AUVB) \leq r(AVUB) \Leftrightarrow C(AUVB) \leq C(AVUB)$

Algorithms

- Cost function
- Query graph
- Join tree shape
- Runtime complexity
- Anytime property
- Result optimality
- When to use which algorithm, why?
- Algorithms composed of subalgorithms

Greedy Heuristics

- Linear (left-deep/right-deep/zig-zag) join tree
  - GJO-1: Start with smallest relation. Pick next relation with smallest cardinality.
  - GJO-2: Start with smallest relation. Pick next relation that minimizes tree cardinality.
  - GJO-3: GJO-2, but try every start relation.

Greedy Operator Ordering (GOO)

- greedily pick edges such that the intermediate result is minimized
- merge nodes connected by the picked edge
- calculate cardinality of merged node
- calculate selectivities of collapsed edges (product of individual selectivities)
- can produce bushy trees

Maximum Value Precedence (MVP)

- heuristic: prefer to perform joins that reduce the input size of expensive operations the most
- algorithm builds an effective spanning tree in the weighted directed join graph (edges and nodes have weights)
  - physical edge: $w_{u,v} = \frac{|\bowtie_u|}{|\mathcal{R}(u) \cap \mathcal{R}(v)|}$
  - virtual edge: $w_{u,v} = 1$
  - node: $w(p_{i,j}, S) = \frac{|\bowtie^S_{p_{i,j}}|}{|R_i \bowtie_{p_{i,j}} R_j|}$
- take edges with weight < 1 (reduce work for later operators)
- add remaining edges (increase input sizes as late as possible)

IKKBZ

- requires acyclic query graph
- generates optimal left deep trees for acyclic queries in polynomial time (requires cost function with ASI property)
- for each relation R in the query graph
  - build the precedence graph rooted in R
  - find subtree whose children are chains
  - build compound relations to eliminate contradictory sequences (normalize)
  - merge chains (ascending in rank)
  - repeat until the whole join tree is a chain
  - denormalize previously normalized compound relations
- pick the cheapest of all generated sequences

Dynamic Programming

- optimality principle
- construct larger trees from optimal smaller ones
- try all combinations that might be optimal
- different possibilities to enumerate sets of relations
  - DP_{size}: enumerate sets ascending in size
  - DP_{sub}: enumerate in integer order
  - DP_{size} vs DP_{sub}: Neither dominates the other! When do we use which?
  - DP_{ccp}: enumerate connected component complement pairs
    - adapts to the shape of the query graph
    - lower bound for all DP algorithms
  - DP_{hyp}: handles hypergraphs (join predicates between more than two relations, reordering constraints for non inner joins, graph simplification)

Adaptive Optimization

- DP optimal but too slow for large queries
- Small: DP
- Medium: LinDP (MST + IKKBZ + linearized DP)
- Huge: GOO/DP

MST

- Kruskal's algorithm: Union find
- (Prim's algorithm)

Memoization

- recursive top-down approach
- memoize already generated trees to avoid duplicate work
- might be faster, as more knowledge allows for more pruning
- usually slower than DP

Transformative Approaches

- apply equivalences to initial join tree
- makes it easy to add new equivalences/rules (in theory)
- use memoization (keep all trees generated so far)
- naive implementation generates a massive amount of duplicates
- duplicates can be avoided by disabling certain rules after a transformation has been applied (introduction of new rules becomes harder)

Permutations

- construct permutations of relations (left deep trees)
- choose each relation as start relation once
  - successively add a relation to the existing chain (recursively enlarge the prefix)
  - only explore the resulting chain further if exchanging the last two relations does not result in a cheaper chain
  - recursion base: all relations are contained in the chain $\Rightarrow$ keep chain if cheaper than cheapest chain seen so far
- any time algorithm (can be stopped as soon as the first complete permutation is generated)
- finds the optimal plan eventually

Random Join Trees (uniformly distributed, with cross products)

general approach:

- set of alternatives S
- count number of alternatives $n = |S|$
- bijection $rank : S \rightarrow [0, n[$
- draw a random number $r \in [0, n[$
- $rank^{-1}(r)$ gives a random element from S (unranking)

implementation

- random permutation (left deep tree, leaf labeling)
- random tree shape (Dyck words)
- random trees without cross products for tree queries (pretty complex)

Quick Pick

- generate pseudo random trees
- randomly pick an edge from the query graph
- no longer uniformly distributed $\Rightarrow$ no guarantees
- use union-find datastructure to identify subsets containing the nodes connected by an edge

Meta Heuristics

- universal optimization strategies
- Iterative Improvement
  - start with random join tree
  - apply random transformation until minimum is reached
  - might be stuck in local minimum
- Simulated Annealing (inspired by metallurgy)
  - start with random join tree
  - apply random transformation
  - accept transformed tree either if it is cheaper or - with a temperature dependent probability - even if it is more expensive
  - decrease temperature over time
  - allows to escape local minima

Meta Heuristics

- Tabu Search
  - start with random join tree
  - investigate cheapest neighbor even if it is more expensive
  - keep (recently) investigated solutions in tabu set to avoid running into circles
- Genetic algorithms
  - population of random join trees
  - use different tree encoding schemes
  - simulate crossover and mutation
  - survival of the fittest

Combinations and Hybrid Approaches

- Two Phase Optimization: II followed by SA
- AB Algorithms: IKKBZ followed by II
- Toured Simulated Annealing: run n times with different initial join trees (e.g. results of GreedyJoinOrdering-3)
- GOO-II: Run II on the result of GOO
- Iterative DP (IDP-1): build join trees with up to k relations, replace cheapest with compound, repeat

Order Preserving Joins

- non-commutative operators
- how to parenthesize the chain?
- maintain arrays cardinalities, costs, trees/split positions

Accessing the Data

- Yao, Cheung: estimate the number of pages to be read from disk if we want to read k (distinct) tuples directly
- Bitvector: estimate sequential disk access costs for a sequence of k pages sorted in the order they reside on disk
- Selectivity estimation (Histograms)
- Random sample

Unnesting

- Eliminate dependent joins by transforming them into normal joins (with additional predicates)
- The attributes of the left input must be propagated up through the right input
- May need to join right input with the domain

Physical properties

- Physical properties (ordering, grouping) of tuples can be exploited by operators to reduce the cost of a plan
- Minimal cost subplan does not necessarily lead to minimal cost plan for entire query
- Need to consider physical properties
- Limit orderings to *interesting* orderings
- Init, test, apply FD: Encode in FSM