Algoritmos para joins óptimos en el peor caso

Clase 26

IIC 3413

Prof. Cristian Riveros

Programa entero para cota de cubrimiento

- $Q \ := \ R_1(\bar{x}_1), \ldots, R_n(\bar{x}_n)$ con $y_1, \ldots, y_m$ todas las variables en Q.
- $\mathcal{H}_Q = (V, E)$ es el hipergrafo con $V = \{y_1, \ldots, y_m\}$ y $E = \{R_1, \ldots, R_n\}$.
- D una base de datos tal que $N_R = |R(D)|$ para todo $R \in E$.

$$|Q(D)| \ \leq \ \min_{C \text{ cubrimiento de } \mathcal{H}_Q} \left\{ \prod_{R \in C} N_R \right\}$$

Cota de cubrimiento versión en programación entera

$$\begin{array}{llll}
\mathcal{P}_{Q,D}: & \text{min} & \displaystyle\prod_{R \in E} (N_R)^{c_R} & \\
& & & \\
& \text{tal que:} & \displaystyle\sum_{R : y \in R} c_R \ \geq \ 1 & \text{para cada variable } y \in V \\
& & c_R \in \{0, 1\} & \text{para cada relación } R \in E
\end{array}$$

Cota AGM (Atserias-Grohe-Marx)

Podemos relajar el programa anterior desde los enteros a **los racionales**:

$$
\begin{array}{llll}
\mathcal{P}^*_{Q,D}: & \min & \displaystyle\sum_{R \in E} \log_2(N_R) \cdot c_R & \\
& \text{tal que:} & \displaystyle\sum_{R : y \in R} c_R \;\geq\; 1 & \text{para cada variable } y \in V \\
& & 0 \leq c_R \leq 1 & \text{para cada relación } R \in E
\end{array}
$$

Teorem (Cota AGM)

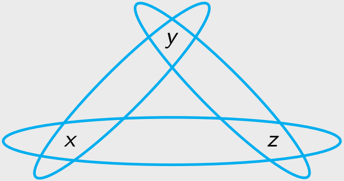
Para toda consulta conjuntiva Q y base de datos D, si $O^*_{Q,D}$ es **el valor óptimo para el programa lineal** $\mathcal{P}^*_{Q,D}$, entonces:

$$|Q(D)| \;\leq\; 2^{O^*_{Q,D}}$$

y existen BD D **arbitrariamente grandes** tal que $|Q(D)| = 2^{O^*_{Q,D}}$.

La cota $2^{O^*_{Q,D}}$ es **óptima***

Conclusión sobre la cota AGM



$$
\begin{array}{lrcl}
\text{min} & N^{c_R} \cdot N^{c_S} \cdot N^{c_T} & & \\
\text{tal que:} & c_R + c_T & \geq & 1 \\
& c_R + c_S & \geq & 1 \\
& c_S + c_T & \geq & 1 \\
& c_R, c_S, c_T & \in & [0, 1]
\end{array}
$$

Conclusión sobre $Q_\Delta := R(x, y), S(y, z), T(z, x)$

- El tamaño de $|Q_\Delta(D)|$ es a lo más $N^{\frac{3}{2}}$.
- El tamaño de $|R \bowtie S|$, $|R \bowtie T|$, o $|T \bowtie S|$ puede ser N^2.

¿es posible calcular $Q_\Delta(D)$ en tiempo $\mathcal{O}(N^{\frac{3}{2}})$?
Para toda Q y D, ¿es posible calcular $Q(D)$ en tiempo a lo más $\mathcal{O}(2^{O^*_{Q,D}})$?

Algoritmos de joins **óptimos en el peor caso**

Algoritmos que calculan $Q(D)$ en tiempo **a lo más** $\mathcal{O}(2^{O^*_{Q,D}})$.

Varias propuestas

- Worst-case optimal join algorithms (2012)
  Hung Ngo, Ely Porat, Christopher Ré, Atri Rudra.
- Beyond worst-case analysis for joins with minesweeper (2014)
  Hung Ngo, Dung Nguyen, Christopher Ré, Atri Rudra.
- Joins via Geometric Resolutions: Worst-case and Beyond (2015)
  Mahmoud Khamis, Hung Ngo, Christopher Ré, Atri Rudra.
- **Leapfrog Triejoin**: A Simple Worst-Case Optimal Join Algorithm (2014) *Todd L. Veldhuizen.*
- ...

Outline

Joins unarios

Leapfrog Triejoin

Outline

Joins unarios

Leapfrog Triejoin

Joins unarios / intersecciones

Suponga una consulta conjuntiva **unaria** (intersección):

$$Q(x) \ := \ R_0(x), \ldots, R_{n-1}(x)$$

¿qué dice la cota AGM sobre el tamaño de $Q(x)$?
¿cómo calculamos $Q(D)$ en tiempo a lo más $\mathcal{O}(\min_i |R_i|)$?

Joins unarios / intersecciones

Suponga una consulta conjuntiva **unaria** (intersección):

$$Q(x) \;:=\; R_0(x), \ldots, R_{n-1}(x)$$

Cada relación R_i esta ordenada de manera **creciente** con la interfaz:

$R_i.\texttt{begin}()$:	posición en el menor valor (default $\perp$).
$R_i.\texttt{key}()$:	retorna el valor actual.
$R_i.\texttt{next}()$:	avanza al siguiente valor mayor al actual.
$R_i.\texttt{seek}(k)$:	avanza al menor valor mayor o igual a k.

- Métodos entregan `null` en caso de llegar al final.
- `begin`, `key`, `next` toman tiempo constante ($\mathcal{O}(1)$).
- `seek`: toman tiempo $\mathcal{O}(\log(|R_i|))$.

¿cómo podemos implementar esta interfaz para cada R_i?

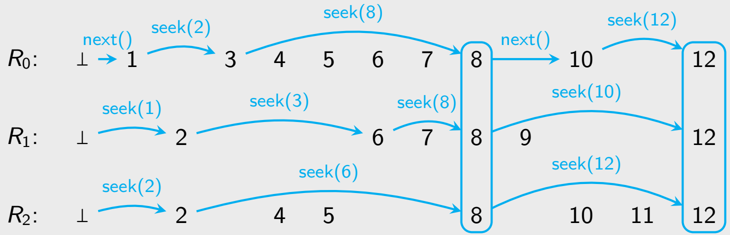
Joins unarios / intersecciones

Cada relación R_i esta ordenada de manera **creciente** con la interfaz:

$R_i.\texttt{begin}()$:	posición en el menor valor $\perp$.
$R_i.\texttt{key}()$:	retorna el valor actual.
$R_i.\texttt{next}()$:	avanza al siguiente valor mayor al actual.
$R_i.\texttt{seek}(k)$:	avanza al menor valor mayor o igual a k.

¿cómo calculamos $Q(x) := R_0(x), \ldots, R_{n-1}(x)$?

Intuición



Algoritmo Leapfrog para joins unarios

Algoritmo

```
open-LF(R_0, ..., R_{n-1})
    for i = 0 ... n-1 do
        R_i.begin()

next-LF(R_0, ..., R_{n-1})
    R_0.next()
    i := 1 mod n
    while true do
        if R_i.key() = R_[(i-1) mod n].key() then
            return R_i.key()
        else
            R_i.seek(R_[(i-1) mod n].key())
            i := (i + 1) mod n
```

Algoritmo Leapfrog para joins unarios

Algoritmo

```
next-LF(R_0, ..., R_{n-1})
    R_0.next()
    i := 1 mod n
    while true do
        if R_i.key() = R_[(i-1) mod n].key() then
            return R_i.key()
        else
            R_i.seek(R_[(i-1) mod n].key())
            i := (i + 1) mod n
```

Tiempo algoritmo Leapfrog

$$\mathcal{O}\left(n \cdot \left(\min_i |R_i|\right) \cdot \log(\max_i |R_i|)\right)$$

Leapfrog caso favorable

Tiempo algoritmo Leapfrog

$$\mathcal{O}\left(n \cdot \left(\min_i |R_i|\right) \cdot \log(\max_i |R_i|)\right)$$

Caso favorable

$$\begin{aligned} R_0 &= \{1, \ldots, 2n\} \\ R_1 &= \{n+1, \ldots, 3n\} \\ R_2 &= \{1, \ldots n, 2n+1, \ldots, 3n\} \end{aligned}$$

- $R_0 \cap R_1 \cap R_2 = \varnothing$
- $R_0 \cap R_1 \neq \varnothing,\ R_1 \cap R_2 \neq \varnothing,\ R_0 \cap R_2 \neq \varnothing$
- Leapfrog toma tiempo $\mathcal{O}(1)$ para todo n.

Un algoritmo de **join de a pares** tomará por lo menos $\Omega(n)$.

Outline

Joins unarios

Leapfrog Triejoin

Orden de variables

Para una consulta conjuntiva (sin proyección ni constantes):

$$Q(y_1, \ldots, y_m) := R_1(\bar{x}_1), R_2(\bar{x}_2), \ldots, R_n(\bar{x}_n)$$

fije el orden de variables $y_1, \ldots, y_m$ (GAO - General Attribute Order)
tal que cada $R_i(\bar{x}_i)$ las variables $\bar{x}_i$ siguen el orden $y_1, \ldots, y_m$.

Ejemplo

$$R(x, y), S(y, z), T(z, x)$$

$$\begin{aligned}
\text{Orden } x, y, z \quad &\Rightarrow \quad R(x, y), S(y, z), T(x, z) \\
\text{Orden } y, z, x \quad &\Rightarrow \quad R(y, x), S(y, z), T(z, x) \\
\text{Orden } z, y, x \quad &\Rightarrow \quad R(y, x), S(z, y), T(z, x)
\end{aligned}$$

Orden de variables

Para una consulta conjuntiva (sin proyección ni constantes):

$$Q(y_1, \ldots, y_m) := R_1(\bar{x}_1), R_2(\bar{x}_2), \ldots, R_n(\bar{x}_n)$$

fije el orden de variables $y_1, \ldots, y_m$ (GAO - General Attribute Order) tal que cada $R_i(\bar{x}_i)$ las variables $\bar{x}_i$ siguen el orden $y_1, \ldots, y_m$.

Definiciones

Para valores $a_1, \ldots, a_{k-1}$ y $R(x_1, \ldots, x_l)$ sea:

$$R[a_1, \ldots, a_{k-1}, y_k] \quad := \quad \pi_{y_k}\big(\sigma_{y_1 = a_1 \wedge \ldots \wedge y_{k-1} = a_{k-1}}(R)\big)$$

Casos bordes:

- si $y_i \notin \{x_1, \ldots, x_l\}$ para $i < k$, entonces $y_i = a_i$ es verdadero.
- si $y_k \notin \{x_1, \ldots, x_l\}$, entonces se define $R[a_1, \ldots, a_{k-1}, y_k] = \texttt{true}$.

Orden de variables

Definiciones

Para valores $a_1, \ldots, a_{k-1}$ y $R(x_1, \ldots, x_l)$ sea:

$$R[a_1, \ldots, a_{k-1}, y_k] \quad := \quad \pi_{y_k}\big(\sigma_{y_1 = a_1 \wedge \ldots \wedge y_{k-1} = a_{k-1}}(R)\big)$$

Ejemplo

Para el orden de variables x, y, z, u:

R	x	y	u
	1	3	4
	1	3	5
	1	4	6
	1	4	8
	1	4	9
	1	5	2
	3	5	2

$$\begin{array}{lcl}
R[x] & := & \{1,3\} \\
R[1,y] & := & \{3,4,5\} \\
R[1,4,u] & := & \texttt{true} \\
R[1,4,7,z] & := & \{6,8,9\}
\end{array}$$

Orden de variables

Para una consulta conjuntiva (sin proyección ni constantes):

$$Q(y_1, \ldots, y_m) := R_1(\bar{x}_1), R_2(\bar{x}_2), \ldots, R_n(\bar{x}_n)$$

fije el orden de variables $y_1, \ldots, y_m$ (GAO - General Attribute Order) tal que cada $R_i(\bar{x}_i)$ las variables $\bar{x}_i$ siguen el orden $y_1, \ldots, y_m$.

Definiciones

Para valores $a_1, \ldots, a_{k-1}$ y $R(x_1, \ldots, x_l)$ sea:

$$\begin{aligned} R[a_1, \ldots, a_{k-1}, y_k] &:= \pi_{y_k}\left(\sigma_{y_1=a_1 \wedge \ldots \wedge y_{k-1}=a_{k-1}}(R)\right) \\ Q[a_1, \ldots, a_{k-1}, y_k] &:= R_1[a_1, \ldots, a_{k-1}, y_k], \ldots, R_n[a_1, \ldots, a_{k-1}, y_k] \end{aligned}$$

Notar que $Q[a_1, \ldots, a_{k-1}, y_k]$ es un **join unitario**!

Algoritmo para multijoin: Leapfrog Triejoin

Algoritmo

input : $Q(y_1, \ldots, y_m) = R_1(\bar{x}_1), R_2(\bar{x}_2), \ldots, R_n(\bar{x}_n)$ y base de datos D
output: Enumerar todas las tuplas en $Q(D)$

```
Leapfrog-TrieJoin(Q, D)
    open-LF(Q[y1])
    foreach a1 ∈ next-LF(Q[y1]) do
        open-LF(Q[a1, y2])
        foreach a2 ∈ next-LF(Q[a1, y2]) do
            open-LF(Q[a1, a2, y3])
            foreach a3 ∈ next-LF(Q[a1, a2, y3]) do
                ...
                open-LF(Q[a1, a2, ..., a(m-1), ym])
                foreach am ∈ next-LF(Q[a1, a2, ..., a(m-1), ym]) do
                    enumerate (a1, ..., am)
```

Algoritmo para multijoin: Leapfrog Triejoin

```
Leapfrog-TrieJoin(Q, D)
    open-LF(Q[y1])
    foreach a1 ∈ next-LF(Q[y1]) do
        open-LF(Q[a1, y2])
        foreach a2 ∈ next-LF(Q[a1, y2]) do
            ...
            open-LF(Q[a1, a2, ..., a_{m-1}, ym])
            foreach am ∈ next-LF(Q[a1, a2, ..., a_{m-1}, ym]) do
                enumerate (a1, ..., am)
```

Teorema (Correctitud)

Algoritmo `Leapfrog-Triejoin` enumera todos los resultados de $Q(D)$.

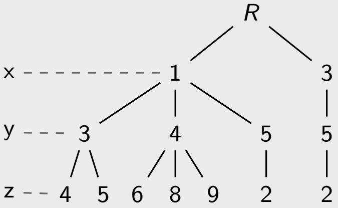
Importante: necesitamos la interfaz de **Leapfrog** para $R[a_1, \ldots, a_{k-1}, y_k]$.

Estructura de trie para relaciones

Para una relación $R(x_1, \ldots, x_k)$ y **orden de variables** $x_1, \ldots, x_n$ contamos con una estructura de **trie** de las tuplas.

Ejemplo

R	x	y	z
	1	3	4
	1	3	5
	1	4	6
	1	4	8
	1	4	9
	1	5	2
	3	5	2



Estructura de trie para relaciones

Para una relación $R(x_1, \ldots, x_k)$ y **orden de variables** $x_1, \ldots, x_n$ contamos con una estructura de **trie** de las tuplas.

Estructurando la relación como un trie, podemos implementar la interfaz:

$R[a_1, \ldots, a_{k-1}, y_k].\texttt{begin}()$:	posición en el menor valor $\perp$.
$R[a_1, \ldots, a_{k-1}, y_k].\texttt{key}()$:	retorna el valor actual.
$R[a_1, \ldots, a_{k-1}, y_k].\texttt{next}()$:	avanza al siguiente valor mayor al actual.
$R[a_1, \ldots, a_{k-1}, y_k].\texttt{seek}(k)$:	avanza al menor valor mayor o igual a k.

para valores $a_1, \ldots, a_{k-1}$ tal que:

- Métodos entregan `null` en caso de llegar al final.
- `key` y `next` toman tiempo constante ($\mathcal{O}(1)$).
- `begin` y `seek`: toman tiempo $\mathcal{O}(\log(|R_i|))$.

¿cómo podemos implementar esta interfaz usando un **trie** (más algo)?

Importante: notar que la estructura depende del **orden GAO**.

Optimalidad Leapfrog Triejoin

Podemos relajar el programa anterior desde los enteros a **los racionales**:

$$\begin{array}{llll}
\mathcal{P}^*_{Q,D}: & \min & \displaystyle\sum_{R \in E} \log_2(N_R) \cdot c_R & \\
& \text{tal que:} & \displaystyle\sum_{R : y \in R} c_R \;\geq\; 1 & \text{para cada variable } y \in V \\
& & 0 \leq c_R \leq 1 & \text{para cada relación } R \in E
\end{array}$$

Teorem (Optimalidad)

Para toda consulta conjuntiva Q y base de datos D, si $O^*_{Q,D}$ es **el valor óptimo para el programa lineal** $\mathcal{P}^*_{Q,D}$, entonces el algoritmo de Leapfrog Triejoin toma tiempo:

$$\mathcal{O}(n \cdot 2^{O^*_{Q,D}} \cdot \log(\max_i |R_i|))$$

FIN . . .

Grupo de investigación en manejo de datos









(Faltan varios)

Temas de investigación

Manejo de datos:

- Big data.
- Datos streaming.
- Extracción de información.

Grafos de datos:

- Web semántica.
- Base de datos de grafos.
- Centralidad de datos.

Lógica / Lenguajes formales:

- Teoría de modelos finitos.
- Teoría de automatas.

Teoría de la Computación:

- Complejidad computacional.
- Algoritmos aleatorios.

Proyectos de implementación

1. MilleniumDB — **Base de datos de grafos**
2. CORE — **Base de datos streaming**
3. REmatch — **Motor de extracción de información**

Estan **invitados a colaborar** en cualquiera de estos proyectos
como práctica laboral (IMFD), IPre, Magister, . . .

(solo escribanme y pregunten)