

Aufgabe 1

Erzeugen Sie (von Hand) die DP-Tabelle, die durch `DPCsgCmp` für die Relationen R_0, R_1, R_2 mit den Kardinalitäten $|R_0| = 10, |R_1| = 2, |R_2| = 100$ und den Selektivitäten $f_{R_0R_1} = 0.5$ und $f_{R_1R_2} = 0.1$ (Kostenfunktion C_{out}) berechnet wird. Markieren Sie den endgültigen Tabelleneintrag, aber geben Sie auch verworfene Einträge an.

Lösung

Prozedur	S_1	Prozedur	S_2	Plan	C_{out}
ECsg	$\{R_2\}$	ECmp	\emptyset		
ECsg	$\{R_1\}$	ECmp	$\{R_2\}$	$R_1 \bowtie R_2$ $R_2 \bowtie R_1$	20 * 20
ECsgRec	$\{R_1, R_2\}$	ECmp	\emptyset		
ECsg	$\{R_0\}$	ECmp	$\{R_1\}$	$R_0 \bowtie R_1$ $R_1 \bowtie R_0$	10 * 10
		ECsgRec	$\{R_1, R_2\}$	$\{R_1, R_2\} \bowtie R_0$ $R_0 \bowtie \{R_1, R_2\}$	120 120
ECsgRec	$\{R_0, R_1\}$	ECmp	$\{R_2\}$	$\{R_0, R_1\} \bowtie R_2$ $R_2 \bowtie \{R_0, R_1\}$	110 * 110
ECsgRec	$\{R_0, R_1, R_2\}$	ECmp	\emptyset		

Aufgabe 2

Für allgemeine Joingraphen (oder auch schon für star queries) benötigt DP exponentielle Laufzeit. Schätzen Sie den Ressourcenaufwand für große Anfragen ab.

Für Star-Queries z.B. werden $(n-1)2^{n-2}$ (Teil-) Pläne gespeichert. Rechnen Sie zur Vereinfachung mit 2^n Plänen.

Aufgabe 2 a)

Wieviele Pläne werden bei einer Anfrage aus n Relationen erzeugt ($n_1 = 20, n_2 = 30$)?

Lösung

- Jeder Plan besteht im Durchschnitt aus $2^{n/2}$ Operationen.
- für $n = 20$: $2^{20} \approx 1$ Million Pläne mit $2^{30} \approx 1$ Milliarde Operationen
- für $n = 30$ sind das $2^{30} = 1$ Milliarde Pläne mit ≈ 35 Billionen Operationen
- dadurch kann schon der Adressraum des Systems erschöpft sein.

Aufgabe 2 b)

Wenn jeder Plan ca. 20 Bytes Hauptspeicher verbraucht, wieviel Hauptspeicher beanspruchen dann alle generierten Pläne für $n = 30$?

Lösung

≈ 21.5 GB

Aufgabe 2 c)

Was kann man bei größeren Problemen machen? (d.h. wie bekommt man eine immer noch brauchbare Lösung)

Lösung

- für Teilprobleme aus z.B. max 20 Relationen lösen
- danach Heuristiken anwenden
- oder diese Teillösungen als Relationen für dynamisches Programmieren benutzen — ggf. auch iterativ.
- Heuristik, z.B. greedy-3, benutzen, um eine Kostenschranke zu berechnen. Die Kostenschranke kann zum Prunen von zu teuren Teillösungen benutzt werden. Dadurch kann Speicherplatz gespart werden.
- Probabilistischer Algorithmus, z.B. Random join tree oder Iterative Improvement

Aufgabe 3

Erweitern Sie das Program vom letzten Übungsblatt, so daß es mit Memoization die optimale Joinreihenfolge findet. Allerdings sollen nur Pläne ohne Kreuzprodukt berücksichtigt werden.

Lösung

siehe MemoizeJoin.java