Aufgabe 9.1:

Lösung:

Sei p die Größe einer Seite, hier $p = 4\ kB$.
Sei b die Größe eines Blocks in Anzahl Seiten, hier $b = 100$.
Die Größe des Datenbankpuffers, d.h. die zur Ausführung des Verbundes im Internspeicher zur Verfügung stehende Anzahl Seiten, sei k; wir nehmen für die folgenden Überlegungen $k = 102$ an. Eine Seite ist hierbei reserviert für die Bildung des Resultates des Verbundes.

Sei l_r die Länge eines Tupels zu R, hier $l_r = 50Byte$ und sei N die Anzahl benötigter Seiten von R, hier $N = 10.000$. Sei n die Anzahl Tupel von R, die sich ergibt zu $n = N * p/l_r$.

Sei l_s die Länge eines Tupels zu S, hier $l_s = 100Byte$ und sei M die Anzahl benötigter Seiten von S, hier $M = 5.000$. Sei m die Anzahl Tupel von S, die sich ergibt zu $m = M * p/l_s$.

Block-Nested-Loop-Verbund Wir wählen die kleinere Relation S als äußere und lesen jeweils immer einen ganzen Block von S-Seiten ein. Insgesamt werden somit M Seiten von S eingelesen und in M/b Iterationen sämtliche Seiten von R nach einander betrachtet. Damit ergeben sich $M + (M/b) * N = 505.000$ Seitenzugriffe zur Berechnung des Verbundes.

Index-Nested-Loop-Verbund Wir nehmen somit einen Index über dem Verbundattribut B der Relation R an. Für jedes Tupel in S wird der Index angefragt. Nehmen wir an, dass wir mit i Seitenzugriffen im Mittel das (die) jeweils korrespondierende(n) Tupel in R lokalisieren können und somit mit $i+1$ Zugriffen in den Datenbankpuffer bringen. Es ergeben sich dann insgesamt $M + m * (i+1) = 405.000$ Seitenzugriffe zur Berechnung des Verbundes, wobei $i = 1$ angenommen wird.

Sort-Merge-Verbund Nach der Lösung zu Aufgabe 8.2 beträgt der Aufwand des externen Sortierens einer Relation R

$$x_R = (\lceil \log_{k-1} \lceil \frac{N}{k} \rceil \rceil + 1) 2 \cdot N,$$

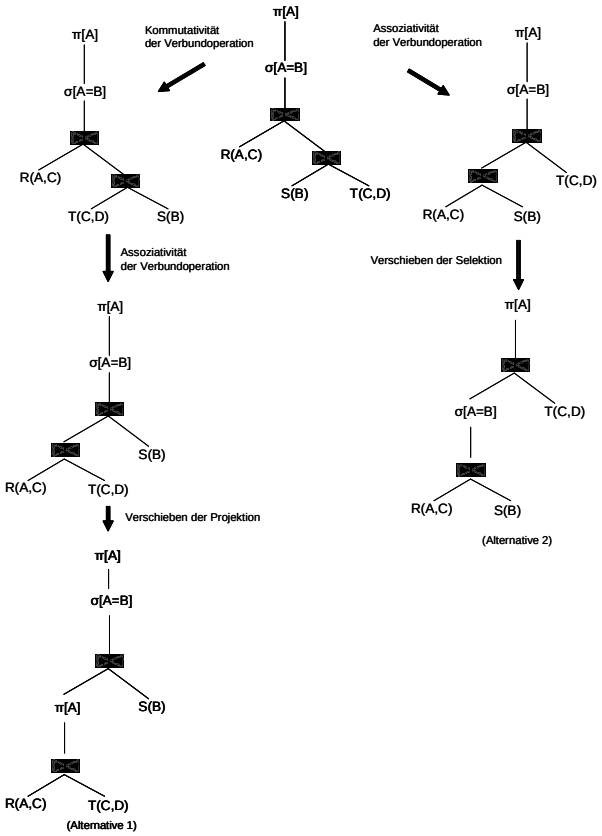
wobei k die Größe des Datenbankpuffers und N die Anzahl Tupel der zu sortierenden Relation. Der Aufwand zur Berechnung des Verbundes ergibt sich damit zu $M + N + x_S + x_R = 75.000$.

Hash-Verbund Zur Anwendung von h_1 auf R und S müssen beide Relationen einmal komplett eingelesen werden. Sei x die Anzahl gebildeter Blöcke der kleineren Relation S. Damit die x Blöcke mit einem Durchlauf über S gebildet werden können, benötigen wir mindestens x Seiten im Datenbankpuffer. Damit das Verfahren effizient angewendet werden kann, muss somit

$x \leq k-1$ gelten. Andererseits muss jeder Block komplett in den Puffer passen, also $M/x \leq k-1$ gelten. Wählen wir $x = k-1$ muss gelten $\sqrt{M}+1 \leq k$. Diese Bedingung ist für obige konkrete Werte erfüllt. Insgesamt ergibt sich somit der Aufwand $2(M+N)$ für die Anwendung von h_1 und $M+N$ für die Anwendung von h_2, insgesamt $3 * (M+N) = 45.000$.

Aufgabe 9.2:

Lösung:



Alternative 1:

1) Berechnung des kartesischen Produkts $S(B) \bowtie T(C, D)$ wird vermieden.
2) Durch Verschieben der Projektion $\pi[A]$ nach unten wird Zwischenergebnis verkleinert.
3) $\sigma[A = B]$ kann für Gleichheitsverbund verwendet werden (kartesisches Produkt braucht nicht berechnet zu werden).

Alternative 2:

1) Durch Gleichheitsverbund $\sigma[A = B](R(A, C) \bowtie (S(B))$ muss kein kartesiches Produkt berechnet werden.

Aufgabe 9.3: (a)

Lösung:

Mit jedem Verbund verdoppelt sich die Anzahl Tupel. Der Ausdruck $\bowtie_{i=1}^{n-1} R_i$ erzeugt somit $8 \cdot 2^{n-2}$ Tupel.

(b)

Lösung:

Ist n ungerade, $n > 1$, dann ist das Ergebnis gerade $\emptyset$. Der Aufwand ist exponentiell in n, da der Ausdruck ein Zwischenergebnis mit $8 \cdot 2^{n-2}$ Tupel erzeugt.

Aufgabe 9.4:

Lösung:

Der innere Ausdruck der Varianten $R \bowtie (S \bowtie T)$ und $(R \bowtie S) \bowtie T$ ist jeweils ein Verbund über zwei Relationen, die zwei gemeinsame Attribute haben. Dies gilt nicht für die dritte Variante $S \bowtie (R \bowtie T)$; hier haben R und T nur ein gemeinsames Attribut. Die ersten beiden Varianten sind somit der dritten vorzuziehen, da, unter gleichen Annahmen über die Wahrscheinlichkeit gleicher Attributwerte, die ersten beiden Varianten zu einem Zwischenergebnis mit weniger Tupeln führen werden als die dritte Variante.

Aufgabe 9.5:

Lösung:

In beiden Repräsentationen werden den Knoten Intervalle zugeordnet. In Kapitel 6.3 ergeben sich die Intervallgrenzen aus den Positionen der Tags in der betrachteten XML-Serialisierung; in Kapitel 9.6 haben wir eine Rangnummerierung für $\texttt{Element}^{opt}$ gewählt, in der der Hauptrang $r_h(k)$ eines Knotens k immer kleiner als sein Nebenrang $r_n(k)$ ist und in der für jeden Knoten k' des Teilbaums mit Wurzel k gerade $r_h(k) < r_h(k') < r_n(k') < r_n(k)$ gilt. Die relativen Verhältnisse der Intervalle unterschiedlicher Knoten sind also in beiden Darstellungen gleich.

Die Techniken aus Kapitel 6.3 und Kapitel 9.6 können somit miteinander kombiniert werden.

Aufgabe 9.6:

Lösung:

In der Tabelle `Element+(Haupt, Neben, Elter, Name)` wird zu jedem Knoten der Hauptrang seines Elterknotens gespeichert. Die Elemente der Achsen `descendant, ancestor, following` und `preceding` können wie im Buch angegeben ohne Referenz auf den Elterknoten identifiziert werden.

Ebenfalls ohne Referenz auf den Elterknoten können die Elemente der folgenden Achsen identifiziert werden:

v' ist Element der `self`-Achse von $v \iff pre(v') = pre(v)$,

v' ist Element der `descendant-or-self`-Achse von $v \iff$
$pre(v) \leq pre(v')$ und $post(v) \geq post(v')$,

v' ist Element der `ancestor-or-self`-Achse von $v \iff$
$pre(v) \geq pre(v')$ und $post(v) \leq post(v')$.

Die folgenden Achsen benötigen zur Identifizierung ihrer Elemente die Referenz auf den Elterknoten. Der Wert des `Elter`-Attributes eines Knotens v' sei gerade $par(v')$, wobei für v Elter von v' gerade $par(v') = pre(v)$.

v' ist Element der `child`-Achse von $v \iff$
$pre(v) < pre(v')$, $post(v) > post(v')$ und $par(v') = pre(v)$,

v' ist Element der `parent`-Achse von $v \iff$
$pre(v) > pre(v')$, $post(v) < post(v')$ und $par(v) = pre(v')$,

v' ist Element der `following-sibling`-Achse von $v \iff$
$pre(v') > pre(v)$, $post(v) < post(v')$ und $par(v) = par(v')$,

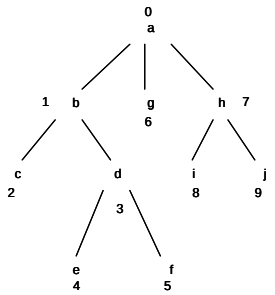
v' ist Element der `preceding-sibling`-Achse von $v \iff$
$pre(v') < pre(v)$, $post(v) < post(v')$ und $par(v) = par(v')$.

Die `attribute`-Achse ergibt sich so wie die `child`-Achse, jedoch muss zusätzlich der Knotentyp getestet werden. Hierzu müssen wir die Tabelle `Element` um eine weitere Spalte `Typ` erweitern, deren Werte `elem` für Element-Knoten und `attr` für Attributknoten sind.

Aufgabe 9.7:

Lösung: (vergl. Grust, Sakr und Teubner, 2004)

Betrachte den XML-Baum mit Knotennummerierung gemäß der Hauptreihenfolge zusammen mit der entsprechende `Element`-Tabelle mit den Angaben der Anzahl Knoten des betreffenden Teilbaumes und der Tiefe des Knotens.



Element

Haupt	Anzahl	Tiefe	Name
0	9	0	a
1	4	1	b
2	0	2	c
3	2	2	d
4	0	3	e
5	0	3	f
6	0	1	g
7	2	1	h
8	0	2	i
9	0	2	j

v' ist Element der `descendant`-Achse von $v \iff$
$pre(v) < pre(v')$ und $pre(v') \leq pre(v) + size(v)$,

v' ist Element der `ancestor`-Achse von $v \iff$
$pre(v) > pre(v')$ und $pre(v) < pre(v') + size(v')$,

v' ist Element der `preceding`-Achse von $v \iff pre(v') + size(v') < pre(v)$,

v' ist Element der `following`-Achse von $v \iff pre(v') > pre(v) + size(v)$,

v' ist Element der `self`-Achse von $v \iff pre(v') = pre(v)$,

v' ist Element der `descendant-or-self`-Achse von $v \iff$
$pre(v) \leq pre(v')$ und $pre(v') \leq pre(v) + size(v)$,

v' ist Element der `ancestor-or-self`-Achse von $v \iff$
$pre(v) \geq pre(v')$ und $pre(v) < pre(v') + size(v')$,

v' ist Element der `child`-Achse von $v \iff$
$pre(v) < pre(v')$, $pre(v') \leq pre(v) + size(v)$ und $level(v') = level(v) + 1$,

v' ist Element der `parent`-Achse von $v \iff$
$pre(v) > pre(v')$, $pre(v) < pre(v') + size(v')$ und $level(v) = level(v') + 1$,

v' ist Element der `following-sibling`-Achse von $v \iff$
$pre(v') > pre(v) + size(v)$ und es existiert ein v'' so,
dass v'' sowohl Element der `parent`-Achse von v als auch von v',

v' ist Element der `preceding-sibling`-Achse von $v \iff$
$pre(v') + size(v') < pre(v)$ und es existiert ein v'' so,
dass v'' sowohl Element der `parent`-Achse von v als auch von v'.