PA152: Efficient Use of DB

8. Summary of Algorithm Costs and Limits

Vlastislav Dohnal

Costs of One-Pass Algorithms

Algorithm	Costs I/O	Relation Size	Memory Blocks	Pipelining
Distinct	B(R)	B(R) ≤ M-2	1-input, 1-output, M-2 search table	yes
Group by	B(R)	B(R) ≤ M-1	1-input, 0-output, M-1 aggregates	no
Set union	B(R)+B(S)	B(R)+B(S) ≤ M-2	1-input, 1-output, M-2 search table	yes
Set intersection	B(R)+B(S)	min(B(R),B(S)) ≤ M-2	1-input, 1-output, M-2 search table	yes, after reading S
Set diff (R-S)	B(R)+B(S)	B(R)+B(S) ≤ M-2	1-input, 1-output, M-2 search table	yes, after reading S
Set diff (S-R)	B(R)+B(S)	B(S) ≤ M-1	1-input, 0-output, M-1 search table	no
Bag union	B(R)+B(S)	unlimited	1-input, 1-output	yes
Bag intersection	B(R)+B(S)	B(S) ≤ M-2	1-input, 1-output, M-2 search table	yes, after reading S
Bag diff (R-S)	B(R)+B(S)	B(S) ≤ M-2	1-input, 1-output, M-2 search table	yes, after reading S
Bag diff (S-R)	B(R)+B(S)	B(S) ≤ M-1	1-input, 0-output, M-1 search table	no
Cross join	B(R)+B(S)	B(S) ≤ M-2	1-input, 1-output, M-2 cache for S	yes, after reading S
(any) Join	B(R)+B(S)	B(S) ≤ M-2	1-input, 1-output, M-2 search table	yes, after reading S

* S is always the smaller relation.

Costs of Join Algorithms

* S is always the smaller relation.
** Y are common attributes.
*** 1.5-pass algorithms

Algorithm	Costs I/O	Relation Size	Memory Blocks	Pipelining
Block-based Nested-loop ***	B(S)·(1+B(R))	unlimited	2-input, 1-output	yes
Cached BB NL ***	B(S)/(M-2) · (M-2 + B(R))	unlimited	1-input, 1-output, M-2 cache of S	yes
Merge Join (w/o sorting)	B(R)+B(S)	unlimited	2-input, 1-output (+x when too many matches)	yes
Merge Join (incl. sorting)	5·(B(R) + B(S))	B(R) ≤ M·(M-1) + 1	<u>sorting</u>: M-input for a run, 0-output <u>merging</u>: (M-1)-runs, 1-output <u>joining</u>: 2-input, 1-output (+x when too many matches)	yes, after sorting R&S
Sort Join	3·(B(R) + B(S))	$M = \left\lceil \frac{B(R)}{M} \right\rceil + \left\lceil \frac{B(S)}{M} \right\rceil + 1$ approx. B(R)+B(S) ≤ M·(M-1)	<u>sorting</u>: M-input for a run, 0-output <u>joining</u>: (M-1)-runs, 1-output (+x when too many matches)	yes, after sorting R&S
Index Join (R.Y index) (max costs)	B(S) + T(S) · (HT + θ) e.g., θ = T(R)/V(R,Y)	unlimited	2-input, 1-output (+x for index cache)	yes

Costs of Join Algorithms

Algorithm	Costs I/O	Relation Size	Memory Blocks	Pipelining
Hash Join	3·(B(R) + B(S))	B(S) ≤ (M-2)·(M-1)	hashing: 1-input, M-1-buckets joining: 1-bucket of R, 1-output, M-2-a bucket of S	yes, after hashing R&S
Hybrid HJ ***	$3(B(R)+B(S)) - \frac{2(B(R)+B(S))}{\left\lceil\sqrt{B(R)}\right\rceil}$	$B(S) \ll M^2$ $M = \frac{B(R)}{\left\lceil\sqrt{B(R)}\right\rceil} + \left(\left\lceil\sqrt{B(R)}\right\rceil\right) + 1$	hashing: 1-input, x=$\left\lceil\sqrt{B(S)}\right\rceil$-1st bucket of S, M-1-x-buckets joining: 1-bucket of R, x-bucket of S, 1-output	yes, after hashing S
Pointer HJ	B(S)+B(R)+T(R) · θ e.g., θ = T(S)/V(S,Y)	"unlimited", hash index on S.Y + pointers must fit in M	indexing: 1-input, M-3-hash index on S joining: 1-block of R, 1-block of S, 1-output, M-3-hash index on S	yes, after indexing S

* S is always the smaller relation.
** Y are common attributes.
*** keeping just 1 bucket of S in memory

Costs of Two-Pass Algorithms

Algorithm	Costs I/O	Relation Size	Memory Blocks	Pipelining
Joins – see slides above				
Distinct (sorting)	3·B(R)	B(R) ≤ M·(M-1) + 1	<u>sorting</u>: M-input for a run, 0-output <u>distinct</u>: (M-1)-runs, 1-output	yes, after initial sorting
Distinct (hashing)	3·B(R)	B(R) ≤ (M-1)·(M-1)	<u>hashing</u>: 1-input, M-1-buckets <u>distinct</u>: M-1-bucket, 1-output	yes, after hashing
Group by (sorting)	3·B(R)	B(R) ≤ M·(M-2)	<u>sorting</u>: M-input for a run, 0-output <u>group by</u>: (M-2)-runs, 1-output, 1-aggregates	yes, after initial sorting
Group by (hashing)	3·B(R)		<u>hashing</u>: 1-input, M-1-buckets <u>group by</u>: M-2-bucket, 1-output, 1-aggregates	yes, after hashing

* S is always the smaller relation.

Costs of Two-Pass Algorithms

Algorithm	Costs I/O	Relation Size	Memory Blocks	Pipelining
Set union (sorting)	3·(B(R)+B(S))	$M = \left\lceil \frac{B(R)}{M} \right\rceil + \left\lceil \frac{B(S)}{M} \right\rceil + 1$ approx. B(R)+B(S) ≤ M·(M-1)	<u>sorting</u>: M-input for a run, 0-output <u>union</u>: M-1-all runs, 1-output	yes, after initial sorting
Set union (hashing)	3·(B(R)+B(S))	B(S) ≤ (M-2)·(M-1)	<u>hashing</u>: 1-input, M-1-buckets <u>union</u>: 1-buckets of R, 1-output, M-2-bucket of S	yes, after hashing
Set/Bag ∩, - (sorting)	3·(B(R)+B(S))	$M = \left\lceil \frac{B(R)}{M} \right\rceil + \left\lceil \frac{B(S)}{M} \right\rceil + 1$ approx. B(R)+B(S) ≤ M·(M-1)	<u>sorting</u>: M-input for a run, 0-output <u>oper</u>: M-1-all runs, 1-output, (+1 for counts)	yes, after initial sorting
Set/Bag ∩,S-R (hashing)	3·(B(R)+B(S))	B(S) ≤ (M-2)·(M-1)	<u>hashing</u>: 1-input, M-1-buckets <u>oper</u>: 1-buckets of R, 1-output, M-2-bucket of S	yes, after hashing R
Set/Bag R-S (hashing)	3·(B(R)+B(S))	B(R) ≤ (M-2)·(M-1)	<u>hashing</u>: 1-input, M-1-buckets <u>diff</u>: 1-buckets of S, 1-output, M-2-bucket of R	yes, after hashing

* S is always the smaller relation.