Vyhodnocování dotazů
slajdy k přednášce NDBI001

Jaroslav Pokorný
MFF UK, Praha
pokorny@ksi.mff.cuni.cz

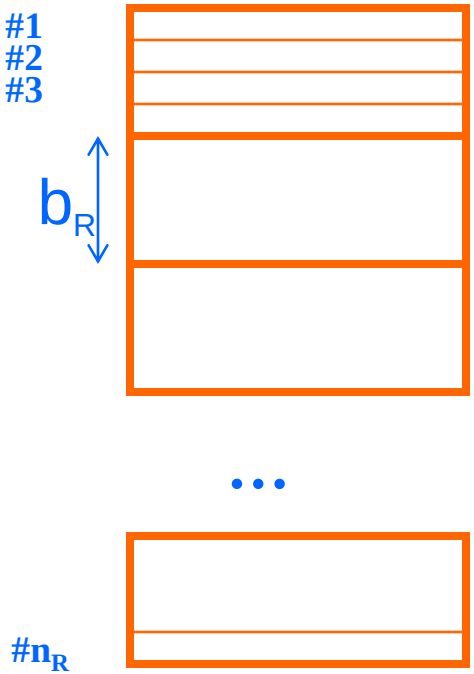
Časová a prostorová složitost

- Jako dlouho trvá dotaz?
  - CPU (cena je malá; snižuje se; těžko odhadnutelná)
  - Disk (hlavní složka ceny, # I/O operací)
- Kolik n-tic je třeba přenést?
- Jaké statistiky je třeba udržovat?

Statistiky

Statistiky pro každou relaci:

n_R	počet n-tic relace R
V(A,R)	počet prvků R[A]
p_R	počet stránek k uložení R
b_R	blokovací faktor
M	počet stránek volného prostoru v RAM
l(A,R)	počet úrovní indexového souboru pro A v R



Značení:

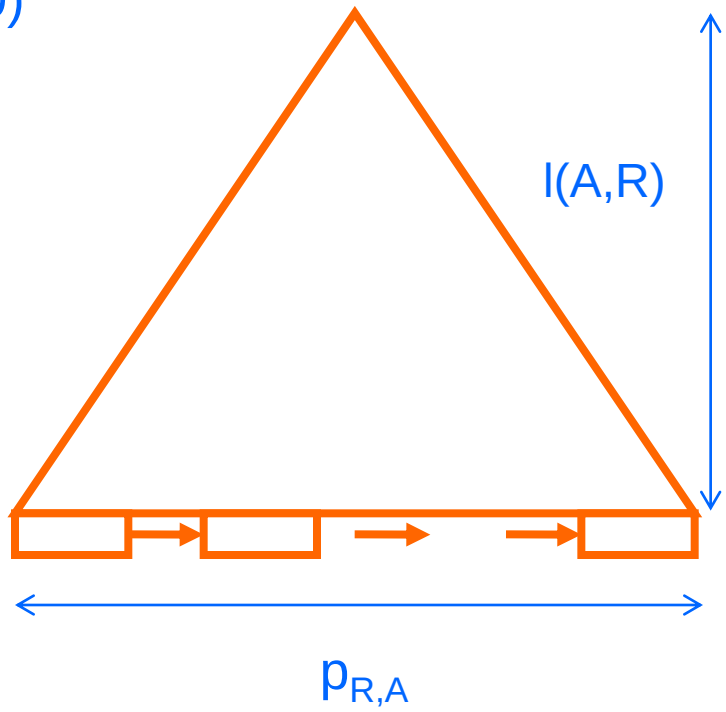
$buffer_R$	celistvý počet stránek pro R v RAM (neuvažujeme caching)

Indexace B^+-stromy:

Pro atribut A relace R:

- $f_{A,R}$: průměrný počet následníků ve vnitřním uzlu (~50-100)
- I(A,R): # úrovní indexu (~2-3)
  - ~ $\log(V(A,R))/\log(f_{A,R})$
- $p_{R,A}$: # listových stránek

Značení: A místo R.A



Metody pro výpočet selekce

select *
from R
where A = ‘a’

Případy: A je primární klíč,
A je sekundární (alternativní) klíč
k A existuje index - obyčejný nebo typu CLUSTER
A je hašovaný klíč

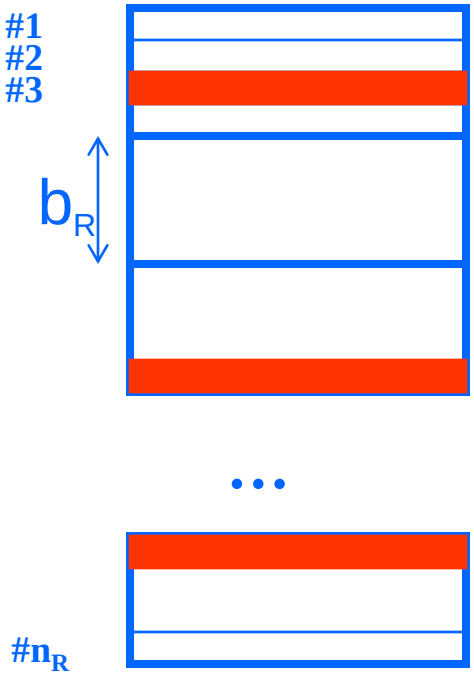
Předpoklady: rovnoměrné rozložení hodnot A v R[A]

$\Rightarrow$ $n_{R(A=a)} = n_R / V(A,R)$

Metody pro výpočet selekce

Sekvenční vyhledávání

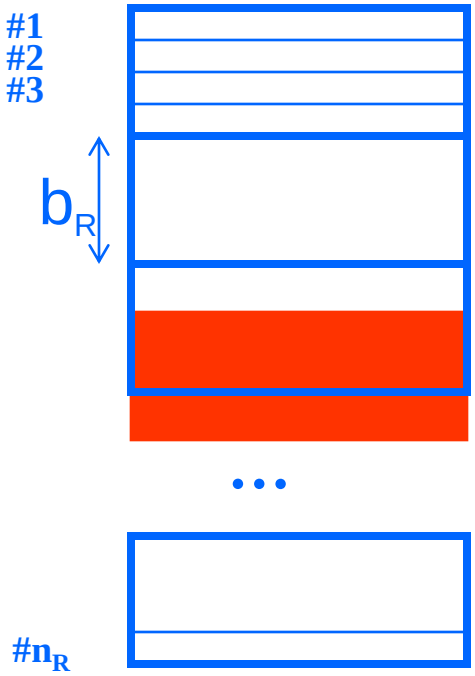
- p_R /*nejhorší případ*/
- $p_R/2$ /* průměrně, je-li A primární klíč*/



Metody pro výpočet selekce

Binární vyhledávání, je-li R uspořádaná podle A

- $\log_2(p_{\mathbf{R}})$ /*je-li A primární klíč */
- $\log_2(p_{\mathbf{R}}) + n_{R(A=a)} / b_{\mathbf{R}}$ /*je-li A libovolný atribut */



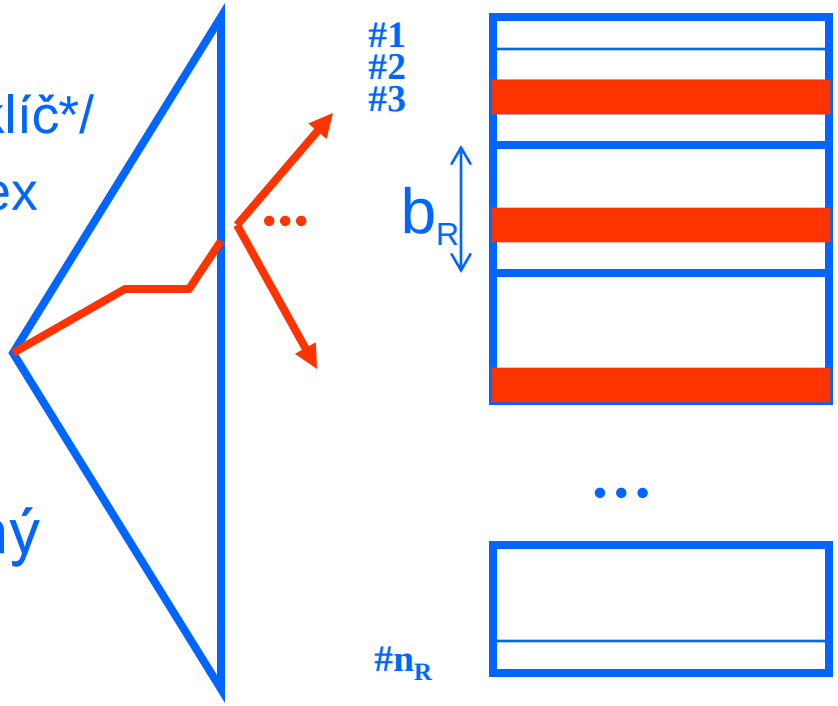
Metody pro výpočet selekce

Vyhledávání, existuje-li pro A index

- $l(A) + 1$ /*je-li A primární klíč*/
- $l(A) + n_{R(A=a)}/b_R$ /*je-li index pro A typu CLUSTER*/
- $l(A) + n_{R(A=a)}$ /*není-li index pro A typu CLUSTER*/

Vyhledávání, je-li A hašovaný atribut

- ≈1 přístup



Metody pro výpočet selekce

select *
from R
where A < ‘a’

Sekvenční vyhledávání

- $p_{\mathbf{R}}$ /* nejhorší případ*/
- $p_{\mathbf{R}}(a - min_A)/(max_A - min_A)$ /*je-li R setříděná dle A*/

Vyhledávání, existuje-li index

- $I(A) + p_{\mathbf{R}}/2$ /*je-li R setříděná dle A*/
- $I(A) + p_{R,A}/2 + n_{\mathbf{R}}/2$ /* je-li index pro A, A je sekundární klíč*/

Příklad

Rezervace(<u>č_zák, č_letu,</u> datum, pozn)

$n_{Rezervace} = 10\ 000$

$b_{Rezervace} = 20$

V(č_zák, Rezervace) = 500

V(č_letu, Rezervace) = 50

$f_{č_letu,R} = 20$

Dotaz: Najdi zákazníky s číslem letu = ‘77’

Příklad

Sekvenční hledání:

$\Rightarrow$ cena dotazu: 500 diskových přístupů

Klastrovaný index pro č_letu:

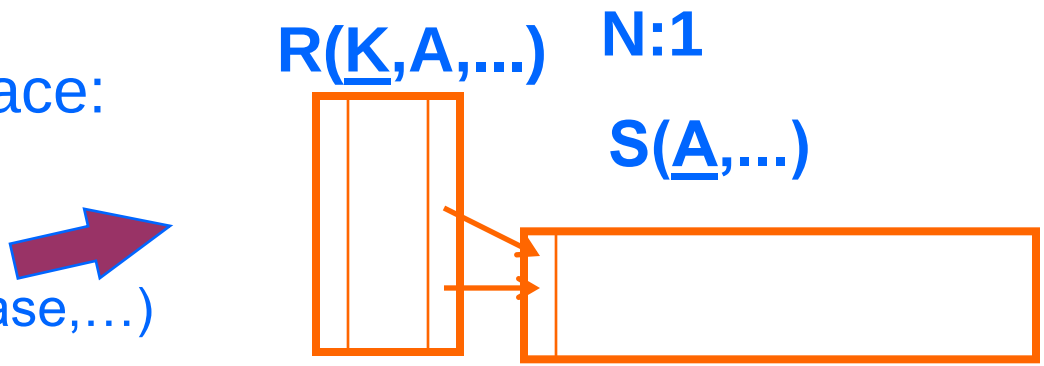
cena dotazu = $I(č_letu) + n_{R(č_letu=a)}/b_{\mathbf{R}}$

- I(A): 50 hodnot $f_A = 20 \Rightarrow I(A)=2$
  Zdůvodnění: $(\log(50)/\log(20) \approx 2$
- $n_{R(A=a)} = n_R/V(A,r) = 10{,}000/50 = 200$ n-tic
  $n_{R(A=a)}/b_{\mathbf{R}} = 200/20 = 10$ stránek
  $\Rightarrow$ cena dotazu = 2+10= 12

Metody pro výpočet spojení

Dva typy implementace:

- nezávislé relace
- s ukazateli (Starbrust, Winbase,…)



Základní metody:

- hnízděné cykly (varianty s indexováním, vyhledáváním)
- setřídění-slévání
- hašované spojení

Předpoklady: spojovací atribut A, $p_R \leq p_S$,

u varianty s ukazateli $R \rightarrow S$

Pz.: speciální případ - kartézský součin

Hnízděné cykly - binárně

- naivní algoritmus

  for each r $\in$ R

  for each s $\in$ S

  if Θ(r,s) then begin u:= r [Θ] s; WRITE(u) end

  …

- po stránkách

  menší relaci jako vnější!

  M=3 $\Rightarrow$ $p_R + p_R p_S$ čtení

  $(n_R\, n_S/V(A,S))/b_{RS}$ zápisů (zdůvodni !)

Vylepšení: - vnitřní relace se čte $\rightleftarrows$

ušetří se 1 čtení na začátku (konci)

Hnízděné cykly - binárně

Varianty:

- M velké, pak rozdělení: M-2, 1, 1

  vnější vnitřní výsledek

  $\Rightarrow p_R + p_S p_R/(M-2)$ čtení
- R ve vnitřní paměti

  $\Rightarrow p_R + p_S$ čtení
- s ukazateli, M=3

  $\Rightarrow p_R + n_R$ čtení

Hnízděné cykly - binárně

- index na S.A (B^+-strom)

Předpoklady: R setříděná v R.A, S.A primární

$\Rightarrow p_R + I(A,S) + p_{S,A} + V(A,R)$ čtení

- S hašovaná dle S.A

Předpoklady: R setříděná v R.A, S.A primární

$\Rightarrow p_R + V(A,R)$ čtení

- se selekcí (pomocí vyhledávání),

Př.: SELECT * FROM R,S

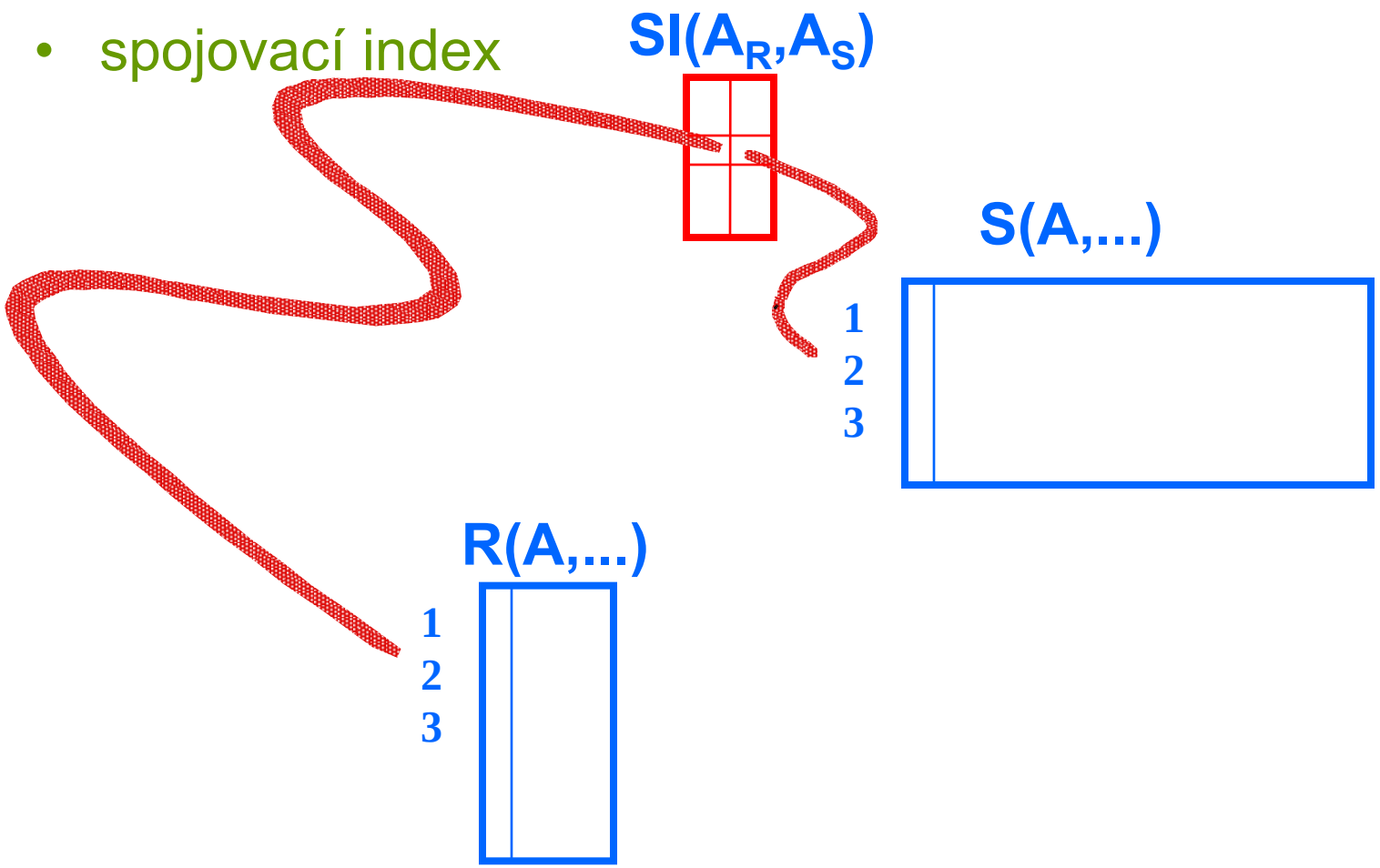
WHERE R.A=S.A AND R.B=12

Předpoklady: R.B primární klíč (indexovaný), S.A sekundární klíč (klastr. Index, n-tice s S.A=a v jedné stránce)

$\Rightarrow I(A,S) + I(B,R) + 2$ čtení

Hnízděné cykly - binárně

- spojovací index



Hnízděné cykly - více relací

$M' = M_1 + M_2 + \ldots + M_n < M$

R_i rozděleny na l_i podrelací velikostí M_i, tj. $l_i = \lceil p_i/M_i \rceil$, $(1 \leq i \leq n)$

cenová funkce [Kim84]

$C = p_1 + [M_2 + (p_2 - M_2)\lceil p_1/M_1 \rceil] + \ldots + [M_n + (p_n - M_n)\lceil p_1/M_1 \rceil \ldots \lceil p_{n-1}/M_{n-1} \rceil]$

$\Rightarrow$ problém hledání celočíselných M_i, aby C minimální

heuristika:

(1) Srovnej n relací do cyklů algoritmu tak, že $p_1 \leq p_2 \leq \ldots \leq p_n$;

(2) Pro R_n alokuj 1 stránku, M' - 1 rozděl rovnoměrně;

(3) (M' - 1)/(n-1) necelé then přiděl větší M_i menším relacím;

Hnízděné cykly - více relací

Struktura základního algoritmu (zde pro tři relace):

```
for j:=1 to L1 do
   begin přečti R1j do bufferM1;
    for k:=1 to L2 do
         begin   přečti R2k do bufferM2;
                 for s:=1 to L3 do
                     begin přečti R3s do bufferM3;
                         vytvoř spojení bufferMi, 1≤i ≤3;
                         zapiš výsledek
                     end
         end
   end
```

Hnízděné cykly - více relací

Př.:

a) $p_1 = 7$, $p_2 = 14$, $p_3 = 21$, $M' = 11$

$\Rightarrow$ dělení $M' = \langle 5, 5, 1 \rangle$

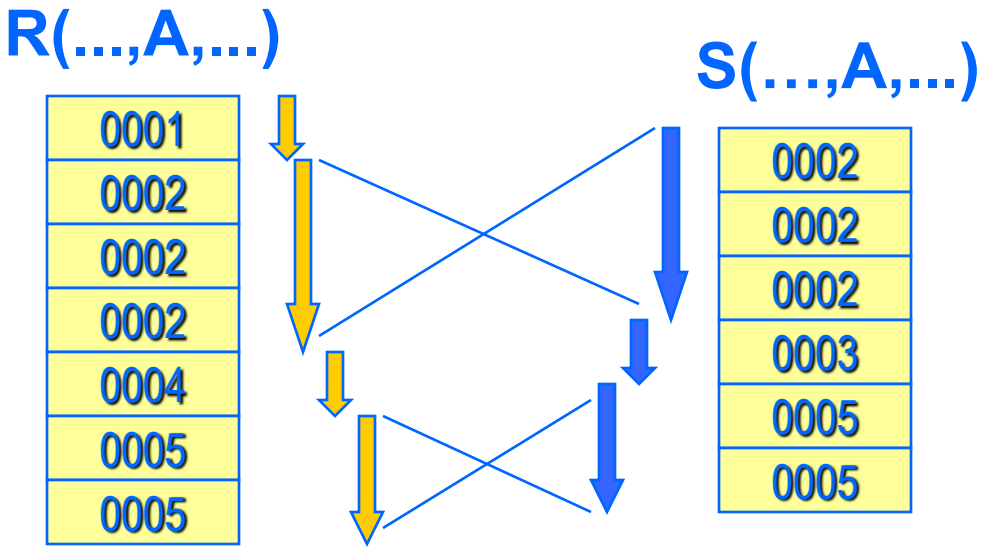
b) $p_1 \leq \ldots \leq p_5$, $M' = 16$

$\Rightarrow$ dělení $M' = \langle 4, 4, 4, 3, 1 \rangle$

Setřídění-slévání

Idea: třídění, slévání (dvoufázový algoritmus)

Vhodnost: jsou-li R a S setříděné



- min. M = 3, 2.fáze vyžaduje $p_R + p_S$ čtení
- potřebuje pomocný prostor pro třídění
- výsledek je setříděný

Setřídění-slévání

- M = 3 (s použitím externího třídění)

  $\Rightarrow \sim 2p_R \log(p_R) + 2p_S \log(p_S) + p_R + p_S$ bez zápisu výsledku
- $M \geq \sqrt{p_S}$ (dvoufázový algoritmus)

(1) Vytvářejí se setříděné běhy velikosti 2M stránek (pomocí prioritní fronty) a ukládají na disk;

$\Rightarrow$ velikost běhu je $\geq 2\sqrt{p_S}$

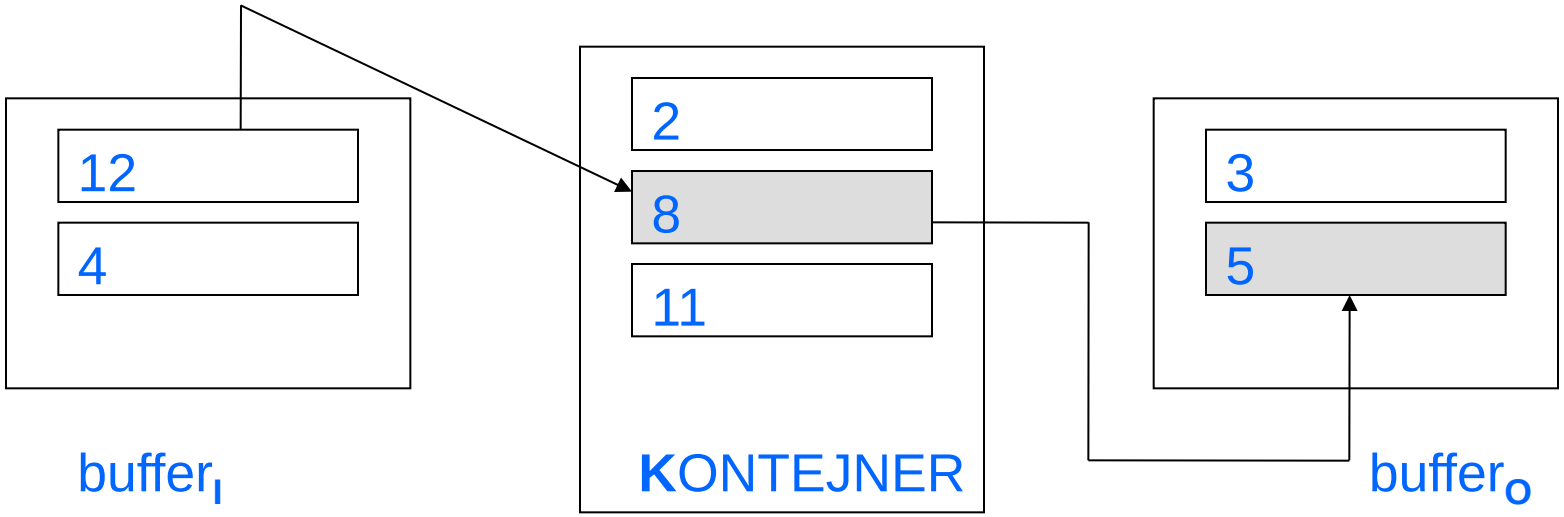
$\Rightarrow$ pro S jich je nejvýše $p_S/2\sqrt{p_S}$,pro R také ne více než $p_S/2\sqrt{p_S}$

$\Rightarrow$ celkem nejvýše $\sqrt{p_S}$

(2) Pro každý běh se alokuje v paměti stránka a souběžně se slévá;

$\Rightarrow 3(p_R + p_S)$ bez zápisu výsledku

Princip prioritní fronty



1. Naplní se K a buffer$_I$ n-ticemi z R.
2. Z K se vybírají n-tice *u* takové, že $\boldsymbol{u.A \geq v.A}$, $\forall v$ ve bufferu$_O$ (1)
   a řadí se ve vzestupném pořadí podle hodnoty *A*.
3. Uvolněné místo v K se zaplní novou n-ticí ze bufferu$_I$. Je-li bufferu$_I$ = ∅, načte se nová stránka R. Je-li buffer$_O$ plný, na vnější paměti se prodlouží daný běh. Jestliže žádná n-tice z kontejneru nevyhovuje (1), je současný stav bufferu$_O$ poslední stránkou vytvářeného běhu.

Tímto způsobem lze vytvořit běhy dlouhé v průměru 2*M* stránek.

Setřídění-slévání

- varianta s ukazateli

R se setřídí podle ukazatelů

S se čte pouze jednou, nemusí být setříděna

$\Rightarrow 3p_R + p_S$ bez zápisu výsledku

Porovnání:

- $|p_R - p_S|$ velké $\Rightarrow$ lepší jsou hnízděné cykly
- $|p_R - p_S|$ malé $\Rightarrow$ lepší je setřídění-slévání
- omezující selekce $\Rightarrow$ lepší pomocí vyhledávání

Hašovaná spojení

Vhodnost:

nejsou-li dostupné indexy pro R.A a S.A

nemusí-li být výsledek setříděn dle A

- klasické hašování
- GRACE algoritmus
- jednoduché hašování
- rekurzivní rozdělení relací
- hybridní hašování

Spojení klasickým hašováním

Předpoklad: R se vejde do M stránek

$M = p_R * F + 1 + 1$, kde F je koeficient větší než 1

(1) Zahašuj R do vnitřní paměti;

(2) Čti S sekvenčně;

Hašuj s.A a přímým přístupem najdi $r \in R$;

(3) if s.A = r.A then begin u:= r * s; WRITE(u) end

$\Rightarrow p_R + p_S$ čtení

Spojení s rozdělováním relací

Předpoklad: R se nevejde do M stránek

Idea: R a S se rozdělí na disjunktní podmnožiny tak, že se spojují *JEN* ty korespondující.

Dvoufázový algoritmus:

(1) Rozděl R a S;

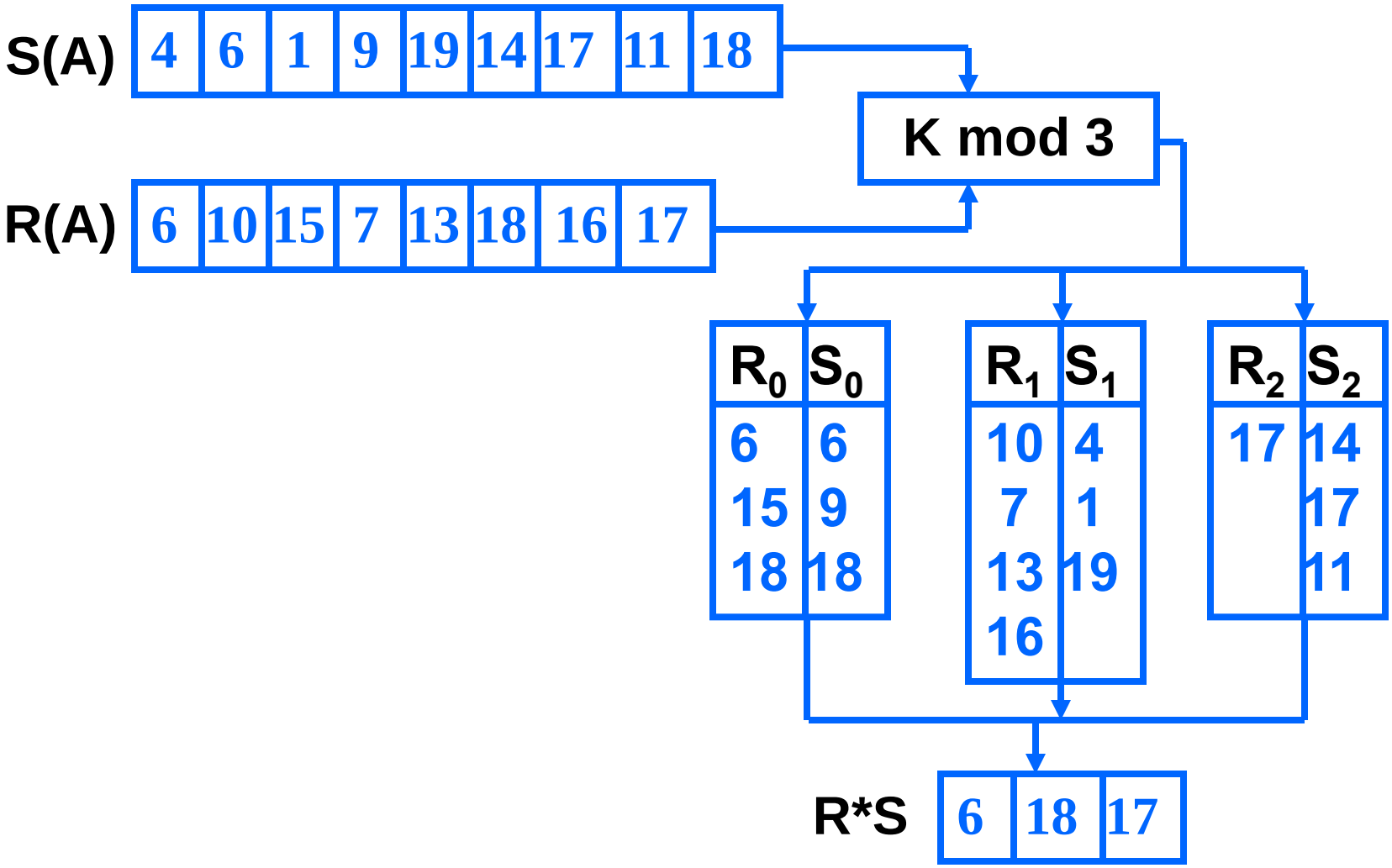
(2) Zahašuj část R (části R) do M-2 stránek;

Čti odpovídající část S;

Hašuj s.A a přímým přístupem najdi n-tice r $\in$ R;

Generuj výsledek;

Příklad:


S(A)
4 6 1 9 19 14 17 11 18
R(A)
6 10 15 7 13 18 16 17
K mod 3
R0 S0
6 6
15 9
18 18
R1 S1
10 4
7 1
13 19
16
R2 S2
17 14
17
11
R*S
6 18 17

GRACE algoritmus

- „školní“ verze

Datové struktury: n-tice R a S, kapsy ukazatelů HR_i, HS_i, $i \in \{0,1,\ldots,m-1\}$

hašovací funkce h: dom(A) $\rightarrow$ <0,m-1>

Algoritmus:

for k:=1 to n_R do begin i :=h(R[k].A); $HR_i := HR_i \cup \{k\}$ end

for k:=1 to n_S do begin i :=h(S[k].A); $HS_i := HS_i \cup \{k\}$ end

for i:=0 to m-1 do

begin $POM_R := \emptyset$; $POM_S := \emptyset$;

foreach $j \in HR_i$ do begin r:=R[j]; $POM_R := POM_R \cup \{r\}$ end;

foreach $j \in HS_i$ do begin s:=S[j]; $POM_S := POM_S \cup \{s\}$ end;

GRACE algoritmus

```
foreach s ∈ POM_S do                                  /* ve vnitřní paměti */
    begin foreach r ∈ POM_R do
            begin if s.A = r.A then begin u:= r * s;
                                                WRITE(u)
                                            end
            end
    end
end
```

$\Rightarrow p_R + p_S + n_R + n_S$ čtení

vhodnost: když $p_R/m + p_S/m < M$

GRACE s ukládáním rozdělených relací

- $M \geq \sqrt{(p_R * F)}$

(1) Zvol h tak, že R lze rozdělit do $m = \sqrt{(p_R * F)}$ částí;

(2) Čti R a hašuj do (výstupních) $buffer_i$, $(0 \leq i \leq m-1)$;

if $buffer_i$ je plný then WRITE($buffer_i$);

(3) Proveď (2) pro S;

(4) for i :=0 to m-1 do begin

(4.1) Čti R_i a hašuj do paměti velikosti $\sqrt{(p_R * F)}$;

(4.2) Čti $s \in S_i$ a hašuj s.A.

Existuje-li $r \in R_i$ a s.A = r.A, generuj výsledek.

end

GRACE s ukládáním rozdělených relací

Zdůvodnění 4.1: předpoklad - $R_i \approx$ stejné velké

$p_R/m = p_R/\sqrt{(p_R*F)} = \sqrt{(p_R/F)}$

R_i vyžaduje prostor $F\sqrt{(p_R/F)} = \sqrt{(p_R*F)}$

$\Rightarrow 3(p_R + p_S)$ I/O operací

vhodnost: když $p_R/m + p_S/m < M$

poznámky:

- S_i mohou být libovolně velké. Vyžadují 1 stránku paměti;
- problém, když V(A,R) je malý;
- vhodné v situacích, když R($\underline{K}$,…), S($\underline{K_1}$,K,…);
- nevejde-li se R_i resp. S_i do M-2 stránek $\Rightarrow$ rekurze
  tj. R_i se rozdělí na $R_{i0}, R_{i1},...,R_{i(k-1)}$ stránek;

Jednoduché hašování

Předpoklad: p_R*F > M-2, A je UNIQUE

Idea: speciální případ GRACE, kdy R $\rightarrow$ R_1, R_2

Algoritmus:

repeat begin zvol h; Čti R a hašuj r.A; M-2 bufferů
tvoří R_1, ostatní n-tice do R_2 na disk;

Čti S a hašuj s.A;

if h(s.A) padne do prostoru R_1

then begin if s.A = r.A then generuj výsledek end

else ulož s do S_2 na disk;

R:=R_2 ; S:= S_2 end

until $R_2 \neq \emptyset$;

Hybridní hašování

Idea: kombinace GRACE a jednoduchého hašování,

R se dělí na části R_1, R_2 ,…, R_k, R_0 tak, že R_0 se vejde do vnitřní paměti.

Rozdělení M-2 stránek: $|buffer_i| = 1$ $(1 \leq i \leq k)$, $|buffer_0| = p_{R0}$

Algoritmus:

(1) Zvol h;

(2) Čti R a hašuj r.A; tvoř R_i $(0 \leq i \leq k)$; /*R_0 je v $buffer_0$*/

(3) Čti S a hašuj s.A; tvoř S_i $(1 \leq i \leq k)$;

if h(s.A) padne do prostoru S_0 then realizuj spojení;

(4) for i:=1 to k do realizuj spojení podle GRACE;

Porovnání algoritmů

Předpoklady:

$M > \sqrt{p_S}$ pro setřídění-slévání

$M > \sqrt{p_R}$ pro hašování

Značení: alg1 >> alg2 $\Leftrightarrow$ alg1 je lepší než alg2

	Setřídění-slévání	GRACE	jednoduché hašování	hybridní hašování
GRACE	>>		>> (pro menší M)	
jednoduché hašování	>>	>> (pro vetší M)		
hybridní hašování	>>	>>	>>	

Dělení

Df.: R a S se schématy Ω_1 resp. $\Omega_2 \subset \Omega_1$

$T = R \div S = R[\Omega_1 - \Omega_2] - ((R[\Omega_1 - \Omega_2] \times S) - R)[\Omega_1 - \Omega_2]$

Př.:

R	A	B
	3	6
	3	2
	8	2
	1	2
	1	3
	3	4
	3	3

S	B
	2
	4
	3

R	A	B
	1	2
	1	3
	3	4
	3	3
	3	6
	3	2
	8	2

po setřídění

T	A
	3

Dělení-hašováním

Idea: vytvoří se kapsy HS_i pro hodnoty z S.B a do nich se ukládají hodnoty z R.A. Hodnoty z $\cap HS_i$ přispívají do výsledku.

Algoritmus: (prvky hašovací tabulky jsou např. typu array nebo set, reprezentují kapsy)

(1) Čti S, spočti h(s.B) a označ prostor (kapsu) $HS_{s.B}$
foreach s.B do $HS_{s.B} := \emptyset$;

(2) for j:=1 to n_R do begin r:=R[j];
if existuje kapsa pro h(r.B)
then $HS_{r.B} := HS_{r.B} \cup \{r.A\}$ end

(3) foreach $HS_{s.B}$ do sort($HS_{s.B}$); /*není nutné*/

(4) Vytvoř $\cap HS_i$ a generuj T;

Další operace

GROUP BY

- index pro A - přes index se získají skupiny
- setřídění podle R.A
- hašováním (jako u dělení)

  foreach a$\in$ R[A] do vytvoř kapsu + proměnná pro výpočet agregační funkce;

DISTINCT

  také pomocí hašování