30. septembra 2000

doc. RNDr. Peter Ružička, CSc

Efektívne paralelné algoritmy

neautorizovaný preklad slajdov
verzia 0.1 build 5
typeset by NoP-LaTeX1.5

preklad: **Peter Hraško**, NoP@hello.to
korektúry: ... zatiaľ bez korektúr
obrázky: ... zatiaľ bez obrázkov
distribúcia: http://www.sturak.sk/~hrasko/download/epa.zip

Predhovor prekladateľa

Zodpovednosť :-)

Dostávate do ruky slovenský preklad anglických slajdov doc. Ružičku, ktoré tento používa ako sprievodný materiál pri prednáškach, a ktorý aj s týmto zámerom dal do obehu.

Autor prekladu sa pri preklade rozhodol pokúsiť sa dať týmto slajdom štýl samostatných ucelených skrípt, čo však miestami zrejme nebolo celkom uskutočniteľné vzhľadom k nekonzistentnosti pôvodných slajdov (čo človek nakoniec u slajdov očakávať ani nemôže). Preto autor vrelo odporúča používať tento preklad ako sprievodný text k prednáške.

Po konzultácii s doc. Ružičkom som nakoniec ešte nútený vložiť sem nasledujúci text, a síce, že

tento preklad je neautorizovaný

... čo znamená, že doc. Ružička **nezodpovedá** za korektnosť údajov v tomto preklade uvedených.

Korektúry

Korektúry, ktoré boli doteraz (30. septembra 2000) v tomto dokumente prevedené, sú zatiaľ len výsledkom "učenia sa" autora prekladu na skúšku z EPA. Ostatné korektúry ponecháva autor prekladu na budúcich čitateľov. Zodpovednosť za akceptáciu čitateľmi zaslaných korektúr nesie aktuálny distribútor tohto prekladu uvedený na titulnej strane. Upozornenia na prípadné chyby v texte posielajte prosím jemu.

Obrázky

Obrázky momentálne nie sú. Človek, ktorý sem tie obrázky mal nakresliť, nakreslil len prvých päť, a aj tie absolútne odflákol ... vyzeralo to tu značne nechutne **nekonzistentne** a **neestiticky** ... tak ani tých prvých päť tu teraz nie je.

Ak by teda niekto mal záujem na prekreslení obrázkov z pôvodných slajdov doc. Ružičku, tak dotyčný musí splniť nasledovné kritériá:

(1) Obrázok musí byť vektorový, nesmie to byť bitmapa.

(2) Obrázok smie mať rozmery max. 10×4 cm.

(3) Použitý font **musí** byť čo najviac podobný štandardnému TEX-ovému fontu Computer Modern Roman, ktorý je použitý aj v tomto dokumente. Autor prekladu odporúča font *Century 731* alebo nejaký iný font z triedy fontov *Century* s podobným vzhľadom. Každopádne, ak sa nádejný "kreslič" vyzná do typografie, potom by to mal byť *statický serifový font.*

(4) Použitý font môže mať veľkosť max. 12pt, odporúčaná a zároveň minimálna veľkosť je 10pt. Formátovanie matematiky **musí** byť rovnaké ako v LATEX-u.

(5) Texty by mali byť (pokiaľ to nie je vyslovene nemožné) zarovnávané podľa y-ovej súradnice, a celé rozloženie textov by malo "lahodiť oku".

(6) Pre čiary treba použiť čo najmenšiu hrúbku. Ideálne sú “hairlines” používané napr. v CorelDRAW.

(7) Podľa možnosti nepoužívať vyplňované plochy. Avšak ak je to vhodné, možno použiť odtiene šedej, no netreba zabúdať na čitateľnosť obrázku po vytlačení aj na ihličkových tlačiarňach.

Nože a vidly

Ak sa niekomu nepáči tento preklad, autor prekladu odporúča dotyčným osobám odložiť nože a vidly a ísť si to hodiť. Autor prekladu sa nepodujal na tento projekt kvôli blahu všeobecných matfyzáckych más, ale robil to hlavne kvôli sebe a svojim najbližším kolegom.

S pozdravom, autor prekladu

Peter Hraško

Obsah

1. Paralelné modely

Algoritmické modely, ktoré môžu byť použité ako všeobecné systémy pre popis a analýzu paralelných algoritmov. Požiadavky (konfliktujúce):

- Jednoduchosť
  - popísať paralelný algoritmus ľahko
  - analyzovať miery výkonu (rýchlosť, komunikácia)
  - byť natoľko hardwarovo nezávislý, ako je to možné
- Implementovateľnosť
  - byť ľahko implementovateľný na paralelných počítačoch
  - analýza by mala vyjadrovať aktuálny výkon algoritmov na paralelných počítačoch

Žiadny jednotlivý model sa neukázal byť dosť dobrý pre väčšinu výskumov v paralelnom spracúvaní. Hojné množstvo špecifických architektúr a paralelných strojov, ale málo zjednoducujúcich techník a metód v litearúre.

1.1 Orientované acyklické grafy

Zavedieme označenie DAG pre orientovaný acyklický graf. Nech V_G (alebo len V) je *množina vrcholov* (*uzlov*) DAGu G a E_G (alebo len E) je *množina hrán* (*liniek*) DAGu G.

Sú vhodné pre analýzu numerických výpočtov. Algoritmus je reprezentovaný triednou DAGov $\{G_n\}_i\}$ s veľkosťou vstupu n. Implementácia algoritmu môže byť špecifikovaná naplánovaním každého uzla na procesor.

Nech je daných p procesorov. S každým vnútorným uzlom asociujeme dvojicu $(j_i, t-i)$; $j_i \leq p$ je index procesora; t_i je časová jednotka.

(1) Ak $t_i = t_k$ pre $i \neq k$, potom $j_i \neq j_k$. (Každý procesor môže vykonať jedinú operáciu v jednej časovej jednotke.)

(2) Ak (i,k) je hrana v DAGu, potom $t_k \geq t_i + 1$. (Operácia reprezentovaná uzlom k by mala byť plánovaná po tom, ako sa vykonala operácia reprezentovaná uzlom i.)

Čas t_i vstupu i je 0. *Plánovanie* je $\{(j_i, t_i) \mid i \in V\}$ používajúce p procesorov. *Čas* je daný ako $\max_{i \in V} t_i$. *Paralelná zložitosť* DAGu je

$$T_p = \min\{\max_{i \in V} t_i\},$$

kde minimum je vzaté zo všetkých plánovaní používajúcich p procesorov.

Príklad 1.1.1: Nech A, B sú $n \times n$ matice. Uvažujme štandardný algoritmus na vypočítanie $C = AB$. Každé C_{ij} je vypočítané podľa

$$C_{ij} = \sum_{\ell=1}^{n} A_{i\ell} B_{\ell j}.$$

S n^3 procesormi môžu byť operácie naplánované po úrovniach, s n procesormi na výpočet každého C_{ij}; teda DAG môže byť naplánovaný na výpočet C v čase $O(\log n)$. ♠

1.2 Shared-memory model

Dva základné módy:

- synchrónny – SIMD (PRAM)
- asynchrónny – MIMD

Príklad 1.2.1: (Násobenie matice a vektora na Shared-Memory modeli)
Nech A je $n \times n$ matica, x je vektor rádu n a oba sú uložené v zdieľanej pamäti; $p \leq n$, $r = \left\lfloor \frac{n}{p} \right\rfloor$; uvažujeme asynchrónny mód. Nech

$$A = \begin{pmatrix} A_1 \\ \vdots \\ A_p \end{pmatrix},$$

kde A_i je veľkosti $r \times n$. Problém teraz je vypočítať $y = Ax$. ♠

Algoritmus 1.2.1:

vstup: $n \times n$ matica A, vektor x rádu n, uložené v zdieľanej pamäti. Inicializované lokálne premenné sú: rád n, číslo procesora i, počet procesorov p ($p \leq n$) a $r = \left\lfloor \frac{n}{p} \right\rfloor$.

výstup: Komponenty $(i-1)r+1, ..., ir$ vektora $y = Ax$ uložené v zdieľanej premennej y.

`begin`

1: `global read` (x, z);
2: `global read` $(A((i-1)r+1 : ir, 1 : n), B)$;
3: `compute` $w = Bz$;
4: `global write` $(w, y((i-1)r+1 : ir))$;

`end`.

Concurrent read premennej x je požadované všetkými procesormi v kroku 1. *Exclusive write* do toho istého pamäťového miesta.

krok 3: $O(\frac{n^2}{p})$ aritmetických operácií

kroky 1+2: $O(\frac{n^2}{p})$ prenosov

krok 4 $\left\lfloor \frac{n}{p} \right\rfloor$ uložení

Dôležitá črta algoritmu: procesory sa nepotrebujú synchronizovať. Na druhej strane, ak $A = (A_1, ..., A_p)$, $x = (x_1, ..., x_p)$, potom $y = A_1x_1 + ... + A_px_p$. Ale výpočet sumy $z_1 + ... + z_p$, $z_i = A_ix_i$ musí byť explicitne synchronizovaný.

PRAMy:

- EREW PRAM – exclusive read, exclusive write
- CREW PRAM – concurrent read, exclusive write
- CRCW PRAM – concurrent read, concurrent write
  - *common* (obyčajný) CRCW – všetky procesory zapisujú rovnakú hodnotu
  - *arbitrary* (ľubovoľný, svojvoľný) CRCW – ľubovoľný procesor uspeje
  - *priority* (prioritný) CRCW – uspeje procesor s minimálnym indexom

pozn.: Zvyčajne píšeme $A := B + C$ namiesto global read (B, x); global read (C, y); set $z := x + y$; global write (z, A);

Algoritmus 1.2.2: (Násobenie matíc na PRAM)

vstup: Dve $n \times n$ matice A, B uložené v zdieľanej pamäti, kde $n = 2^k$. Inicializované lokálne premenné sú n a trojica indexov (i, ℓ, l) identifikujúcich procesor.

výstup: Produkt $C = AB$ uložený v zdieľanej pamäti.

begin

1: compute $C'(i, j, \ell) = A(i, j)B(\ell, j)$;

2: for $k = 1$ to $\log n$ do

    if $\ell \leq \frac{n}{2^k}$ then set $C'(i, j, \ell) := C'(i, j, 2\ell - 1) + C'(i, j, 2\ell)$;

3: if $\ell = 1$ then set $C(i, j) := C'(i, j, 1)$;

end.

n^3 procesorov, $O(\log n)$ paralelných krokov; vyžaduje concurrent-read (krok 1) CREW PRAM

Príklad 1.2.2: Máme dané pole A $n = 2^k$ čísiel a PRAM s n procesormi $\{P_1, ..., P_n\}$, počítame $S = A(1) + ... + A(n)$. Každý procesor vykoná rovnaký algoritmus. ♠

Algoritmus 1.2.3:

vstup: Pole A rádu $n = 2^k$ uložené v zdieľanej pamäti PRAMu s n procesormi. Inicializované lokálne premenné sú n a číslo procesora i.

výstup: Suma položiek A uložené v zdieľanej premennej S. Pole A obsahuje svoje pôvodné hodnoty.

begin

1: global read $(A(i), a)$;

2: global write $(a, B(i))$;

3: for k=1 to $\log n$ do

    if $i \leq \frac{n}{2^k}$ then begin

        global read $(B(2i - 1), x)$;
        global read $(B(2i), y)$;
        set $z := x + y$;
        global write $(z, B(i))$;

    end

4: if $i = 1$ then global write (z, S);

end.

1.3 Sieťový model

1.3.1 Lineárne procesorové pole a kruh

Algoritmus 1.3.1: (Aynchrónny súčin matica–vektor na kruhu)

vstup: **(1)** Číslo procesora i **(2)** počet p procesorov **(3)** i-ta podmatica $B = A(1:n,(i-1)r+1:ir)$ veľkosti $n \times r$, kde $r = \left\lfloor \frac{n}{p} \right\rfloor$ **(4)** i-ty podvektor $w = x((i-1)r+1:ir)$ veľkosti r.

výstup: Procesor P_i vypočíta vektor $y = A_1x_1 + ... + A_ix_i$ a pošle výsledok napravo. Keď algoritmus skončí, P_1 bude poznať Ax.

`begin`

1: `Vypočítaj súčin` $z = Bw$ `matice` B `a vektora` w;

2: `if` $i = 1$ `then set` $y := 0$ `else receive` (y, left);

3: `set` $y := y + z$;

4: `send` (y, right);

5: `if` $i = 1$ `then receive` (t, left);

`end`.

1.3.2 Mriežka

Priemer ($p = m^2$)-procesorovej mriežky je $\sqrt{p}$; max. stupeň ľubovoľného uzla je 4; takmer každý netriviálny výpočet vyžaduje $\Omega(\sqrt{p})$ paralelných krokov.

Príklad 1.3.1: (Systolický súčin matíc na mriežke)
Pozri obrázok **??** ♠

Systolické algoritmy operujú plne synchrónnym spôsobom, kde v každej časovej jednotke procesor dostane dáta od nejakých susedov, potom vykoná lokálnu operáciu a nakoniec pošle dáta niektorému zo svojich susedov.

1.3.3 Hyperkocka

Algoritmus 1.3.2: (Suma na hyperkocke)

vstup: Pole A $n = 2^d$ provkov, kde $A(i)$ je uložené v lokálnej pamäti procesora uzla P_i n-procesorovej synchrónnej hyperkocky, kde $0 \leq i \leq n-1$.

výstup: Súčet $S = \sum_{i=0} n - 1A(i)$ uložený v P_0.
Nasledujúci algoritmus je pre procesor P_i.

`begin`

1: `for` $\ell = d - 1$ `to` 0 `do`

`if` $0 \leq i \leq 2^\ell - 1$ `then set` $A(i) := A(i) + A(i^{(\ell)1}$

[1] **pozn.:** $i^{(\ell)}$ je i s negovaným ℓ-tým bitom; INF: $i^{(\ell)} := i$ `xor` (1 `shl` ℓ)

end.

Koniec po $\log n$ paralelných krokoch.

Algoritmus 1.3.3: (Broadcast z jedného procesora po hyperkocke)

vstup: Procesor P_0 $p = 2^d$-procesorovej synchrónnej hyperkocky drží informáciu X vo svojom registri $D(0)$.

výstup: X je vyslané všetkým procesorom tak, že $D(i) = X$, kde $1 \leq i \leq p-1$. Algoritmus je pre procesor P_i.

begin

1: for $\ell = 0$ to $d-1$ do

if $0 \leq i \leq 2^\ell$ then set $D(i^{(\ell)}) := D(i)$

end.

Broadcast potrvá $O(\log n)$ paralelných krokov.

Príklad 1.3.2: (Násobenie matíc na hyperkocke)

$C = AB$ na synchrónnej hyperkocke s $p = n^3$ procesormi, kde C, A, B sú rádu $n \times n$. Nech $n = 2^q$ a teda $p = 2^{3q}$. Indexujeme procesory podľa (ℓ, i, j) tak, že $P_{\ell,i,j}$ reprezentuje P_r, kde $r = \ell n^2 + in + j$. Ak upevníme ľubovoľnú dvojicu indexov ℓ, i, j a budeme meniť ten ostávajúci index, dostaneme podkocku rozmeru q.

A je uložené v $P_{\ell,i,0}$, $0 \leq \ell, i \leq n-1$ tak, že $A(i,\ell)$ je v $P_{\ell,i,0}$.

B je uložené v $P_{\ell,0,j}$, $0 \leq \ell, j \leq n-1$ tak, že $B(\ell,j)$ je v $P_{\ell,0,j}$.

Vypočítame $C(i,j) = \sum_{\ell=0}^{n-1} A(i,\ell)B(\ell,j)$ pre $0 \leq i,j \leq n-1$.

(1) Vstupné dáta sú distribuované tak, že $P_{i,j,\ell}$ obsahuje $A(i,\ell)$ a $B(\ell,j)$ pre $0 \leq \ell, i, j \leq n-1$.

(2) $P_{\ell,i,j}$ vypočíta $C'(\ell,i,j) = A(i,\ell)B(\ell,j)$.

(3) Pre $\forall i,j; 0 \leq i,j \leq n-1$ procesory $P_{\ell,i,j}$ $(0 \leq \ell \leq n-1)$ vypočítajú $C(i,j) = \sum_{\ell=1}^{n-1} C'(\ell,i,j)$.

krok 1 Broadcast $A(i,\ell)$ z $P_{\ell,i,0}$ všetkým $P_{\ell,i,j}$; Broadcast $B(\ell,j)$ z $P_{\ell,0,j}$ všetkým $P_{\ell,i,j}$ — $O(\log n)$ krokov.

krok 2 Vykoná sa 1 násobenie v každom $P_{\ell,i,j}$.

krok 3 Vypočíta sa n^2 súm $C(i,j)$; výrazy (terms) $C'(\ell,i,j)$ každej sumy spočívajú v q-rozmernej hyperkocke $\{P_{\ell,i,j} \mid 0 \leq \ell \leq n-1\}$. Použije sa algoritmus 1.3.2. Každá suma môže byť vypočítaná v $O(\log n)$ paralelných krokoch.

Teda $C = AB$ môže byť vypočítaná v čase $O(\log n)$ na n^3-procesorovej hyperkocke. ♠

1.4 Porovnanie paralelných modelov

1.4.1 Orientované acyklické grafy (DAG model)

Tento model sa používa na špecializovanú triedu problémov; trpí niekoľkými nedostatkami. Predstavuje iba čiastočnú informáciu o paralelnom algoritme (musí byť vyriešený problém plánovania + problém alokácie). Nie je (it is no) prirodzený mechanizmus na zvládnutie komunikácie medzi procesormi a alokácií a prístupu do pamäte.

1.4.2 Shared-memory model

Je najvhodnejší pre všeobecnú prezentáciu paralelných algoritmov.

1.4.3 Sieťový model

Je primeranejší na riešenie výpočtových a komunikačných otázok ako DAG model. Má dva hlavné nedostatky:

(1) Je významne zložitejší na popis a analýzu algoritmov.

(2) Silne závisí na čiastočnej topológii. Rôzne topológie môžu vyžadovať úplne rozdielne algoritmy na vyriešenie toho istého problému (ako v probléme násobenia matíc).

1.4.4 PRAM model

Je najvhodnejší algoritmický model:

- Existujú techniky a metódy na zvládnutie problémov na PRAM.
- PRAM odstraňuje detaily týkajúce sa synchronizácie a komunikácie (dovoľuje zamerať sa na štrukturálne vlastnosti problému.
- PRAM vyjadruje niekoľko dôležitých parametrov paralelných výpočtov (explicitná alokácia procesorov a pochopenie operácií).
- PRAM-návrhové-paradigmy sú robustné. Sieťové algoritmy môžu byť odvodené od PRAM algoritmov. PRAM algoritmy môžu byť zobrazené na siete (ohraničeného stupňa).

1.5 Zložitosť paralelných algoritmov

Je daný problém Q; PRAM algoritmus pre Q v čase $T(n)$ použijúc $P(n)$ procesorov. Súčin Čas-Procesor $C(n) = T(n)P(n)$ je *cena* paralelného algoritmu. Paralelný algoritmus môže byť prevedený na sekvenčný algoritmus bežiaci v čase $O(C(n))$. Použijúc $p \leq P(n)$ procesorov, simulácia skonzumuje celkovo $O(\frac{T(n)P(n)}{p})$ času.
Štyri spôsoby merania zložitosti paralelných algoritmov: (asymptoticky ekvivalentné)

(1) $P(n)$ procesorov a $T(n)$ čas.

(2) $C(n) = P(n)T(n)$ cena a $T(n)$ čas.

(3) $O(\frac{T(n)P(n)}{p})$ čas pre p procesorov; $p \leq P(n)$.

(4) $O(\frac{C(n)}{p} + T(n))$ čas pre ľubovoľný počet p procesorov.

PRAM aloritmus na výpočet sumy n prvkov:

(1) n procesorov a $O(\log n)$ čas.

(2) $O(n \log n)$ cena a $O(\log n)$ čas.

(3) $O(\frac{n \log n}{p})$ čas pre $p \leq n$ procesorov.

(4) $O(\frac{n \log n}{p} + \log n)$ čas pre ľubovoľné p.

PRAM algoritmus pre násobenie matíc:

(1) n^3 procesorov a $O(\log n)$ čas.

(2) $O(n^3 \log n)$ cena a $O(\log n)$ čas.

(3) $O(\frac{n^3 \log n}{p})$ čas pre $p \leq n^3$ procesorov.

(4) $O(\frac{n^3 \log n}{p} + \log n)$ čas pre ľubovoľné p.

1.6 Work-Time prezentačný systém paralelných algoritmov

Work-Time (WT) paradigma poskytuje neformálne vodítko pre dvojúrovňový top-down popis paralelných algoritmov. Vrchná úroveň utajuje detaily algoritmu. Spodná úroveň sleduje všeobecný plánovací princíp v plnom popise PRAMu.

1.6.1 Vrchná úroveň (WT prezentácia algoritmu)

Popisuje algoritmus v pojmoch postupnosti časových jednotiek, kde každá časová jednotka môže zahŕňať ľubovoľný počet konkurentných operácií. *Práca* vykonaná paralelným algoritmom je celkový počet použitých operácií.

Algoritmus 1.6.1: (Suma)

vstup: $n = 2^k$ čísiel uložených v poli A.

výstup: Suma $S = \sum_{i=1}^{n} A(i)$.

begin

1: for $1 \leq i \leq n$ pardo

set $B(i) := A(i)$;

2: for $k = 1$ to $\log n$ do

for $1 \leq i \leq \frac{n}{2^k}$ pardo

set $B(i) := B(2i-1) + B(2i)$;

3: set $S := B(1)$;

end.

$T(n) = \log n + 2 = O(\log n)$; $W(n) = n + \sum_{j=1}^{n} \frac{n}{2^j} + 1 = O(n)$

1.6.2 Spodná vrstva (plánovací princíp WT)

Nech $W_i(n)$ je počet vykonaných operácií v časovej jednotke i, kde $1 \leq i \leq T(n)$. Simulujme každú množinu $W_i(n)$ operácií v najviac $\left\lceil \frac{W_i(n)}{p} \right\rceil$ paralelných krokoch p procesormi, pre každé $1 \leq i \leq T(n)$. Ak je simulácia úspešná, odpovedajúci p-procesorový PRAM použije najviac

$$\sum_i \left\lceil \frac{W_i(n)}{p} \right\rceil \leq \sum_i \left(\left\lfloor \frac{W_i(n)}{p} \right\rfloor + 1 \right) \leq \left\lfloor \frac{W(n)}{p} \right\rfloor + T(n)$$

paralelných krokov, ako sme požadovali.

Algoritmus 1.6.2: (Suma pre procesor P_s)

vstup: Pole A veľkosti $n = 2^k$ uložené v zdieľanej pamäti. Inicializované lokálne premenné sú **(1)** n rád poľa A; **(2)** počet p procesorov, $p = 2^q \leq n$; **(3)** s číslo procesora P_s.

výstup: Suma prvkov A uložená v zdieľanej premennej S. Pole A si ponechá svoje pôvodné hodnoty. Nech $h = \left\lfloor \frac{n}{p} \right\rfloor$.

`begin`

1: `for` $j = 1$ `to` h `do`

`set` $B(k(s-1)+j) := A(k(s-1)+j)l$;

2: `for` $k = 1$ `to` $\log n$ `do`

`if` $h - k - q \geq 0$ `then`

`for` $j = 2^{h-k-q}(s-1)+1$ `to` $2^{h-k-q}s$ `do`

`set` $B(j) := B(2j-1) + B(2j)$

`else`

`if` $s \leq 2^h - k$ `then set` $B(s) := B(2s-1) + B(2s)$;

3: `if` $s = 1$ `then set` $S := B(1)$;

`end`.

$$T_p = O\left(\frac{n}{p} + \sum_{h=1}^{\log n} \left\lceil \frac{n}{2^h p} \right\rceil\right) = O\left(\frac{n}{p} + \log n\right).$$

1.6.3 Predstava optimálnosti

Je daný výpočtový problém Q sekvnečnej časovej komplexnosti $T^*(n)$. Paralelný algoritmus riešiaci Q (daný vo vrchnej WT úrovni) je *optimálny*, ak $W(n) = \Theta(T^*(n))$. *Zrýchlenie* dosiahnuté paralelným algoritmom s p procesormi je $S_p(n) = \frac{T^*(n)}{T_p(n)}$ je čas paralelného algoritmu s p procesormi riešiaci Q. Optimálny paralelný algoritmus v čase $T(n)$ *môže byť simulovaný* na p-procesorovom PRAMe v čase $T_p(n) = O\left(\frac{T^*(n)}{p} + T(n)\right)$ (použitím WT princípu).

$$\text{Zrýchlenie } S_p = \Omega\left(\frac{T^*(n)}{\frac{T^*(n)}{p} + T(n)}\right) = \Omega\left(\frac{pT^*(n)}{T^*(n) + pT(n)}\right)$$

Optimálne zrýchlenie (napr. $S_p = \Theta(p)$) sa dosiahne, keď $p = O\left(\frac{T^*(n)}{T(n)}\right)$.

Optimálny paralelný algoritmus je *WT-optimálny* (optimálny v silnom zmysle), ak môže byť ukázané, že $T(n)$ nemôže byť vylepšené žiadnym iným optimálnym algoritmom.

Príklad 1.6.1: (PRAM algoritmus na výpočet sumy vo WT konštrukcii)

$T(n) = O(\log n)$ a $W(n) = O(n)$. Pretože $T^*(n) = n$, tento algoritmus je optimálny. Získa optimálne zrýchlenie vždy, keď $p = O\left(\frac{n}{\log n}\right)$. ♠

Ľubovoľný CREW PRAM bude potrebovať $\Omega(\log n)$ času na výpočet sumy, bez ohľadu na počet procesorov. *WT-optimálny algoritmus pre* CREW PRAM[2]

[2] pozn.prekl.: ???

2. Základné techniky

2.1 Vyvážené stromy

2.1.1 Optimálny algoritmus prefixových súčtov

Postupnosť n prvkov $\{x_1, ..., x_n\} \subseteq S$ (s binárnou asociatívnou operáciou $*$). *Prefixové súčty* $s_i = x_1 * ... * x_i$, $1 \leq i \leq n$.

Triviálny sekvenčný algoritmus: $s_i = s_i - 1 * x_i$ pre $2 \leq i \leq n$. Zložitosť $O(n)$.

Algoritmus 2.1.1: (Rekurzívne prefixové sumy)

vstup: Pole $n = 2^k$ prvkov $(x_1, ..., x_n)$; k je nezáporné celé.

výstup: Prefixové sumy s_i pre $1 \leq i \leq n$.

begin

1: if $n = 1$ then begin

set $s_1 := x_1$;
halt;

end;

2: for $1 \leq i \leq \frac{n}{2}$ pardo

set $y_i := x_{2i-1} * x_{2i}$;

3: Rekurzívne počítaj prefixové sumy z $\{y_1, ..., y_{\frac{n}{2}}\}$ a ulož ich do $z_1, ..., z_{\frac{n}{2}}$

4: for $1 \leq i \leq n$ pardo

if $i = 1$ then

set $s_1 := x_1$

else if $i > 1$ je párne then

set $s_i := z_{\frac{i}{2}}$

else if $i > 1$ je nepárne then

set $s_i := z_{\frac{i-1}{2}} * x_i$;

end.

Veta 2.1.1: *Algoritmus na prefixové sumy počíta prefixové sumy n prvkov v čase $T(n) = O(\log n)$ používajúc celkovo $W(n) = O(n)$ operácií.*

Dôkaz 2.1.1: Správnosť dokážeme indukciou na k, kde veľkosť vstupu je $n = 2^k$.

1° $k = 0$ krok 1

2° Predpokladajme platnosť pre $n = 2^k$. Podľa indukčného prepokladu premenné $z_1, ..., z_{\frac{n}{2}}$ (krok 3) obsahujú prefixové sumy od $\{y_1, ..., y_{\frac{n}{2}}\}$, kde $y_i = x_{2i-1} * x_{2i}$ a $z_j = y_1 * ... * y_j$ a teda $z_j = x_1 * x_2 * ... * x_{2j}$.

krok 4 $\quad i = 2j: \quad s_i = z_{\frac{i}{2}}$

$\quad i = 2j+1: \quad s_i = s_{2j+1} = s_{2j} * x_{2j+1} = z_{\frac{i-1}{2}} * x_i$

$T(n) = T(\frac{n}{2}) + a; W(n) = W(\frac{n}{2}) + bn.$ EREW PRAM

□

2.1.2 Nerekurzívny algoritmus na prefixové sumy

Nech $A(i) = x_i$, $1 \leq i \leq n$. Nech $B(h,j)$, $C(h,j)$ sú pomocné premenné ($0 \leq h \leq \log n$, $1 \leq j \leq \frac{n}{2^h}$) na zaznamenávanie informácie počas dopredného prechádzania stromu (B) a spätného prechádzania stromu (C).

Algoritmus 2.1.2: (Nerekurzívne prefixové sumy)

vstup: Pole A veľkosti $n = 2^k$, kde k je nezáporné celé.

výstup: Pole C také, že $C(0,j)$ je j-ta prefixová suma, $1 \leq i \leq n$.

```
begin
1: for 1 ≤ j ≤ n pardo
       set B(0,j) := A(j);
2: for h = 1 to log n do
       for 1 ≤ j ≤ n/2^h pardo
           set B(h,j) := B(h-1,2j-1) * B(h-1,2j);
3: for h = log n to 0 do
       for 1 ≤ j ≤ n/2^h pardo
           if j = 1 then
             set C(h,1) := B(h,1)
           else if j > 1 je párne then
             set C(h,j) := C(h+1, j/2)
           else if j > 1 je nepárne then
             set C(h,j) := C(h+1, (j-1)/2) * B(h,j);
end.
```

2.2 Pointer jumping

Zakorenený (orientovaný) strom T je orientovaný strom s koreňom r. Technika *Pointer jumping* dovoľuje rýchle spracovanie dát uložených v zakorenených stromoch.

2.2.1 Nájdenie koreňov lesa

Nech F je les zakorenených stromov špecifikovaný poľom P takým, že $P(i) = j$ práve vtedy, keď $(i, j) \in F$ a $P(i) = i$ pre koreň i. Problém je určiť koreň $S(j)$ stromu obsahujúceho vrchol j.

Jednoduchý sekvenčný algoritmus: identifikovať korene; reverzné hrany; BFS ... čas O(n).

Algoritmus 2.2.1: (Pointer Jumping)

vstup: Les zakorenených stromov, každý so samo-slučkou v koreni také, že každý oblúk je špecifikovaný podľa $(i, P(i))$, $1 \leq i \leq n$.

výstup: Pre $\forall i$: koreň $S(i)$ stromu obsahujúceho i.

```
begin
  1: for 1 ≤ i ≤ n pardo
       begin
         set S(i) := P(i);
         while S(i) ≠ S(S(i)) do set S(i) := S(S(i));
       end
end.
```

$T(n) = O(\log n)$; $W(n) = O(n \log n)$ CREW PRAM

Špeciálny prípad: každý strom je spájaný zoznam. Prefixová suma na spájaných zoznamoch sa nazýva *paralelný prefix.*

2.3 Divide & Conquer

Divide & Conquer (DnC) stratégia pozostáva z troch hlavných krokov:

(1) Rozdelenie vstupu do inekoľkých tried rozkladu približne rovnakých veľkostí.

(2) Rekurzívne riešenie podproblému pre každú triedu rozkladu vstupu.

(3) Skombinovanie alebo spojenie riešení rôznych podproblémov do riešenia celkového problému.

2.3.1 Problém konvexného obalu

Nech $S = \{p_1, ..., p_n\}$ je množina n bodov v rovine, reprezentovaných súradnicami (x, y). *Planárny konvexný obal* množiny S je najmenší konvexný polygón obsahujúci S. Polygón Q je *konvexný*, ak pre dané $p, q \in Q$ priamkový segment, ktorého koncové body sú p, q leží celý v Q. *Problém konvexného obalu* (CHP[1]) je určiť usporiadaný (povedzme v smere hod. ručičiek) zoznam CH(S) bodov S definujúcich hranice konvexného obalu S.

Jednoduchá DnC stratégia rieši CHP v $O(n \log n)$ sekvenčnom čase. Triedenie môže byť redukované na CHP. Teda $T^*(n) = \Theta(n \log n)$.

Pozn.: Utriedenie n čísiel môže byť vykonané v čase $O(\log n)$ na EREW PRAM použitím celkovo $O(n \log n)$ operácií. (Teraz využijeme túto skutočnosť.)

[1] FROM ENGLISH: Convex-Hull Problem

2.3.2 Paralelný algoritmus pre Problém konvexného obalu

Nech $p, q \in S$ sú body s najmenšou a najväčšou x-ovou súradnicou, resp. p, q sú v $CH(S)$ a rozkladajú $CH(S)$ na *vrchný obal* UH(S) a *spodný obal* LH(S). Prepokladajme, že $n = 2^k$ a žiadne dva body nie sú totožné.

Predpríprava: V čase $O(\log n)$ a s $O(n \log n)$ operáciami utriedime body p_i podľa x-ovej súradnice: $x(p_1) < x(p_2) < ... < x(p_n)$. Nech $S_1 = (p_1, ..., p_{\frac{n}{2}})$ a $S_2 = (p_{\frac{n}{2}}, ..., p_n)$. Predpokladajme, že $UH(S_1)$ a $UH(S_2)$ boli určené. *Spoločná vrchná dotyčnica* medzi $UH(S_1)$ a $UH(S_2$ je spoločná dotyčnica taká, že oba $UH(S_1)$ a $UH(S_2)$ sú pod ňou. Horná spoločná dotyčnica môže byť zostrojená v čase $O(\log n)$ (sekvenčne) metódou binárneho vyhľadávania.

Keď sú dané $UH(S_1)$ a $UH(S_2)$, určenie hornej spoločnej dotyčnice medzi nimi: konštrukcia $UH(S)$ v čase $O(1)$ a celkovo $O(n)$ operácií. CREW PRAM

Algoritmus 2.3.1: (Jednoduchý vrchný obal)

vstup: Množina S bodov v rovine (žiadne dva nemajú rovnakú x-ovú alebo y-ovú súradnicu) taká, že $x(p_1) < x(p_2) < ... < x(p_n)$, kde n je mocnina 2.

výstup: Vrchný obal S.

`begin`

1: `if` $n \leq 4$ `then`

`Použi brute-force algoritmus na určenie` $UH(S)$ `a skonči;`

2: `Nech` $S_1 = (p_1, ..., p_{\frac{n}{2}})$ `a` $S_2 = (p_{\frac{n}{2}+1}, ..., p_n)$.
`Rekurzívne počítaj` $UH(S_1)$ `a` $UH(S_2)$ `paralelne;`

3: `Nájdi vrchnú spoločnú dotyčnicu bedzi` $UH(S_1)$ `a` $UH(S_2)$ `a odvoď vrchný obal` S`;`

`end`.

Veta 2.3.1: *Algoritmus správne vypočíta vrchný obal n bodov v rovine. Tento algoritmus zbehne v čase $O(\log^2 n)$ s použitím celkovo $O(n \log n)$ operácií.*

Dôkaz 2.3.1: Jednoduchou indukciou na n s predpokladom správnosti spoločnej vrchnej dotyčnice. □

$$T(n) \leq T\left(\frac{n}{2}\right) + a \log n = O(\log^2 n) W(n) \leq 2W\left(\frac{n}{2}\right) + bn = O(n \log n)$$

Algoritmus potrebuje CREW PRAM (cvičenie: adaptovať na EREW PRAM). V Time-Processor systéme: $T(n) = O\left(\frac{n \log n}{p} + \log^2 n\right)$. Tento algoritmus dosiahne optimálne zrýchlenie pre $p \leq \frac{n}{\log n}$.

2.4 Rozkladanie

Stratégia rozkladania pozostáva z

(1) Rozbitia problému do p nezávislých podproblémov rovnakých veľkostí.
(2) Vyriešenia p podproblémov súčasne.

Spojenie dvoch utriedených postupností:

– S s čiastočným usporiadaním $\leq$

– S s lineárnym usporiadaním $\leq$

Nech $A = (a_1, ..., a_n)$, $B = (b_1, ..., b_n)$ sú dve beklesajúce postupnosti. V nasledujúcich častiach uveieme nejaké sekvenčné algoritmy riešiace problém spájania.

2.4.1 Jednoduchý spájací algoritmus

Nech $X = (x_1, ..., x_n)$ je postupnosť. Nech $x \in S$. $\texttt{rank}(x : X)$ je počet prvkov X menších alebo rovných ako x. Umiestnenie $Y = (y_1, ..., y_s)$ v X: $(\texttt{rank}(Y : X) = (r_1, ..., r_s)$, kde $r_i = \texttt{rank}(y_i, X))$. Prepokladajme, že A, B sú rôzne. *Problém spojenia* je pre všetky $x \in A \cup B$ určiť $\texttt{rank}(x : A \cup B)$.

Vieme riešiť tento problém určením $\texttt{rank}(A : B)$ a $\texttt{rank}(B : A)$. $\texttt{rank}(A : B)$ môže byť paralelne vypočítaný v čase $O(\log n)$. Celkový počet operácií je $O(n \log n)$ — neoptimálne riešenie.

2.4.2 Optimálny spájací algoritmus

Algoritmus 2.4.1: (Rozdelenie)

vstup: Dve polia $A = (a_1, ..., a_n)$, $B = (b_1, ..., b_m)$ v rastúcom usporiadaní, kde aj $\log m$ aj $k(m) = \frac{m}{\log m}$ sú celé.

výstup: $k(m)$ dvojíc (A_i, B_i) podspostupností A a B takých, že **(1)** $|B_i| = \log m$; **(2)** $\sum_i |A_i| = n$; **(3)** každý prvok A_i a B_i je väčší ako každý prvok A_{i-1} a B_{i-1} pre všetky $1 \leq i \leq k(m) - 1$.

begin

1: set $j(0) := 0, j(k(m)) := n$;

2: for $1 \leq i \leq k(m) - 1$ pardo

Umiestni $b_{i \log m}$ v A použijúc binárne vyhľadávanie a nech $j(i) = \texttt{rank}(b_{i \log m} : A)$;

3: for $0 \leq i \leq k(m) - 1$ pardo

begin

set $B_i := (b_{i \log m + 1}, ..., b_{(i+1) \log m})$;
set $A_i := (a_{j(i)+1}, ..., a_{j(i+1)})$;
// A_i je prázdna, ak $j(i) = j(i+1)$

end

end.

Algoritmus rozdelí A a B do dvojice podpostupností (A_i, B_i) takých, že $|B_i| = O(\log m)$, $\sum_i |A_i| = n$. Utriedená postupnosť $C = (C_0, C_1, ...)$, kde C_i sme získali spojením A_i a B_i, jde skonštruovaná v čase $O(\log n)$ použitím celkovo $O(m + n)$ operácií.

krok 1: sekvenčný čas $O(1)$

krok 2: čas $O(\log n)$ a práca je $O(\log n \frac{m}{\log m}) = O(m + n)$, pretože $m \frac{\log m}{\log n} < m \frac{\log(n+m)}{\log m} \leq n + m$ pre $n, m \geq 4$.

krok 3: paralelný čas $O(1)$ použitím lineárneho počtu operácií.

Veta 2.4.1: *Nech A, B sú utriedené postupnosti dĺžky n. Spojenie A a B môže byť vykonané v čase $O(\log n)$ použijúc celkovo $O(n)$ operácií.*

Dôkaz 2.4.1: Uvažujme spájanie (A_i, B_i) pre $|B_i| = \log n$. Ak $|A_i| = O(\log n)$, potom spojíme každé (A_i, B_i) v sekvenčnom čase $O(\log n)$ použitím optimálneho sekvenčného algoritmu.

Inak použijeme algoritmus 2.4.1 na rozdelenie A_i do blokov veľkosti $O(\log n)$ (v tomto prípade A_i hrá úlohu B a B_i úlohu A). Tento krok zoberie $O(\log \log n)$ času použijúc celkovo $O(|A_i|)$ operácií.

Nakoniec, optimálny sekvenčný algoritmus pre spájanie môže byť použitý na každú dvojicu podpostupností.

Krok 2: A sa umiestni do B súčasne. CREW PRAM

□

2.5 Pipelining

2-3 strom je strom, v ktorom každý vnútorný vrchol má práve dvoch alebo práve troch potomkov. Utriedený zoznam $A = (a_1, ..., a_n)$, kde $a_1 < a_2 < ... < a_n$ môže byť reprezentovaný 2-3 stromom tak, že každý vnútorný vrchol v obsahuje $L(v), M(v), R(v)$ – najväčšiu hodnotu v ľavom, strednom a pravom podstrome.

2.5.1 Základné operácie na 2-3 stromoch

Sekvenčné `Search`, `Insert`, `Delete` v čase $O(\log n)$. Vloženie postupnosti k prvkov $b_1 < b_2 < \ldots < b_k$ do 2-3 stromu T by pomocou predchádzajúcich elementárnych sekvenčných operácií potrebovalo $O(k \log n)$ operácií v čase $O(\log n)$.

***Pipelining*:**

- Vložiť b_1 a b_k do T použitím sekvenčného vkladacieho algoritmu v sekvenčnom čase $O(\log n)$.
- Súčasne určiť polohu všetkých b_i v T pomocou vyhľadávacej procedúry v $O(\log n)$ paralelných krokoch použitím celkovo $O(k \log n)$ operácií.
- CREW PRAM; (vyhľadanie všetkých b_i potrebuje concurrent-read)

Cvičenie: Ukázať, ako dosiahnuť rovnaký výkon na EREW PRAM.

Reťaz je usporiadaná množina prvkov b_i, ktoré pasujú medzi a_i a a_{i+1} (vyjadruje B_i (???)). Nech $|B_i| = k_i$, $1 \leq i \leq n-1$. Zreteľne $\sum_i k_i = k - 2$. Špeciálny prípad: $k_i \leq 1$ pre všetky $1 \leq i \leq n-1$.

Jednoduchý paralelný algoritmus na paralelné vkladanie: Súčasne vložiť všetkých k prvkov. Vnútorné vrcholy na výške 1 môžu mať 3..6 potomkov. Teda možno rozdeliť rodičovský vrchol na dva také, že každý bude mať majviac 3 potomkov. Toto možno vykonať súčasne pre všetky vnútorné vrcholy na výške 1 s viac ako 3 potomkami. Toto možno opakovať až po koreň. — $O(\log n)$ času použitím $O(k \log n)$ operácií.

Všeobecný prípad: Uvažujme $|B_i| = k_i$; zoberme stredný prvok B_i a použijeme jednoduchý vkladací algoritmus. Opakujeme tento proces pre nové reťaze. — $O(\log k \log n)$ času.

Pipelining: Každá aplikácia jednoduchého vkladacieho algoritmu sa nazýva *úsek*. Na úsek i sa možno pozerať ako na vlnu vystupujúcu na T_i (???). Rôzne vlny môžu byť *pipelined* jedna za druhou nahor po strome. — $O(\log n + \log k) = O(\log n)$ času a $O(k \log n)$ operácií.

CREW PRAM

2.6 Accelerated Cascading

2.6.1 Počítanie maxima v konštantom čase

Algoritmus 2.6.1: (Základné maximum)

vstup: Pole A p rôznych prvkov.

výstup: `Boolean` pole M také, že $M(i) =$ `true` práve vtedy, keď $A(i)$ je maximum z A.

`begin`

1: `for` $1 \leq i, j \leq p$ `pardo`

`if` $A(i) \geq A(j)$

`then set` $B(i,j) :=$ `true`
`else set` $B(i,j) :=$ `false;`

2: `for` $1 \leq i \leq p$ `pardo`

`set` $M(i) := B(i,1) \wedge B(i,2) \wedge ... \wedge B(i,p)$`;`

`end`.

Maximum z p prvkov možno vypočítať na common CRCW PRAM v čase $O(1)$ použitím celkovo $O(p^2)$ operácií.

2.6.2 Dvoj-logaritmický algoritmus

Úroveň vrchola w je jeho jeho vzdialenosť od koreňa. Prepokladajme, že $n = 2^{2^k}$. *Strom s dvoj-logaritmickou hĺbkou* (DLDT[2]) s n listami je taký, že koreň má $2^{2^{k-1}}$ potomkov (napr. $\sqrt{n}$) a každý z jeho potomkov má $2^{2^{k-2}}$ potomkov, etc. Teda každý vrchol na úrovni i má $2^{2^{k-i-1}}$ potomkov a vrchol na úrovni k má 2 potomkov. Počet vrcholov na úrovni i je $2^{2^k - 2^{k-i}}$ a na úrovni k je $2^{2^{k-1}} = \frac{n}{2}$. Hĺbka takéhoto stromu je $k + 1 = \log\log n + 1$ $(0 \leq i \leq k)$.

DLDT počíta maximum n prvkov v čase $O(\log\log n)$ nasledovne:

- Každý vnútorný vrchol obsahuje maximum prvkov vo svojom podstrome.
- Algoritmus 2.6.1 is using in bottom-up, at any level needs čas $O(1)$.
- Počet operácií na úrovni i je $O((2^{2^{k-i-1}})^2)$ na jeden vrchol $(0 \leq i \leq k)$

  Celkový počet operácií je $O((2^{2^{k-i-1}})^2(2^{2^k - 2^{k-i}})) = O(2^{2^k}) = O(n)$ operácií na jednu úroveň.

  $W(n) = O(n \log\log n)$ nie je optimálny !!

[2] FROM ENGLISH: Doubly Logarithmic-Depth Tree

2.6.3 Rýchly algoritmus spravíme optimálnym

Accelerated cascading:

(1) Začneme s optimálnym algoritmom pokiaľ veľkosť problému nezredukujeme na istú prahovú hodnotu.

Použijeme algoritmus s binárnym stromom, začneme od listov, moving $\lceil \log\log\log n \rceil$ (???)[3], pretože počet kandidátov sa redukuje o $\frac{1}{2}$ na úroveň, maximum je medzi $n' = O(\frac{n}{\log\log n})$ prvkami.

$T(n) = O(\log\log\log n)$; $W(n) = O(n)$.

(2) Potom sa presunieme na rýchly ale neoptimálny algoritmus.

Použijeme DLDT na n' prvkoch.

$T(n) = O(\log\log n') = O(\log\log n)$; $W(n) = O(n'\log\log n') = O(n)$.

2.7 Lámanie symetrie

Nech $G = (V, E)$ je orientovaná kružnica. *k-farbenie* G je zobrazenie $c : V \longrightarrow \{0, 1, ..., k-1\}$ také, že $c(i) \neq c(j)$, ak $(i, j) \in E$. Určíme 3-farbenie G.

2.7.1 Priamočiary algoritmus (sekvenčný)

Traverzujeme cyklus G z ľubovoľného vrchola a použijeme striedavo farby 0 a 1 na susedné vrcholy. Tretia farba môže byť potrebná na zakončenie kružnice ... optimálny sekvenčný algoritmus.

2.7.2 Základný farbiaci algoritmus

Algoritmus 2.7.1: (Základné farbenie)

vstup: Orientovaná kružnica veľkosti n; hrany špecifikované poľom S; vrcholové farbenie c.

výstup: Ďalšie farbenie c' vrcholov kružnice.

```
begin
1: for 1 ≤ i ≤ n pardo
       begin
           set k := pozícia najnižšieho bitu, v ktorom sa c(i) a c(S(i)) líšia;
           set c'(i) := 2k + c(i)_k;
           // c(i)_k je k-ty najnižší (najmenej významný) bit c(i)
           // inak, c(i)_k := (c(i) and (1 shl k)) shr k
       end
end.
```

Veta 2.7.1: *Výstupná funkcia c' generovaná algoritmom 2.7.1 je platné farbenie vždy, keď c je platné farbenie. Algoritmus beží v čase $T(n) = O(1)$ a používa $W(n) = O(n)$ operácií.*

[3] pozn.prekl.: nevedel som prečítať

Dôkaz 2.7.1: Index k musí vždy existovať, pretože c je farbenie.

Ak $c'(i) = c'(j)$, potom $c'(i) = 2k + c(i)_k = c'(j) = 2\ell + c(j)_\ell$; teda $k = \ell$ a $c(i)_k = c(j)_k$, čo je spor. Teda $c'(i) \neq c'(j)$.

Pozície najnižších bitov, v ktorých sa dve binárne čísla líšia možno nájsť v čase $O(1)$ sekvenčne vždy, keď každé z tých dvoch binárnych čísiel je veľkosti $O(\log n)$ bitov. □

2.7.3 Optimálny 3-farbiaci algoritmus

Utriedenie n celých čísiel z intervalu $\langle 0, O(\log n)\rangle$ možno vykonať v čase $O(\log n)$ použitím lineárneho počtu operácií. (Kombinácia algoritmov *radix-sort* a *prefix-sums*).

Algoritmus 2.7.2: (3-farbenie kružnice)

vstup: Orientovaná kružnica dĺžky n, ktorej hrany sú špecifikované poľom S.

výstup: Vrcholové 3-farbenie kružnice.

begin

1: for $1 \leq i \leq n$ pardo

set $c(i) := i$;

2: Použijeme algoritmus 2.7.1 raz;

3: Utriedime vrcholy podľa ich farby;

4: for $i = 3$ to $\lceil \log n \rceil$ do

for všetky vrcholy v farby i pardo

Zafarbi v najmenšou farbou z $\{0, 1, 2\}$ rôznou od farieb jeho dvoch susedov;

end.

Veta 2.7.2: *Môžeme zafarbiť vrcholy orientovanej kružnice tromi farbami v čase $O(\log n)$ použitím celkovo $O(n)$ operácií.* EREW PRAM

Dôkaz 2.7.2:

krok 1+2: $O(1)$ času a $O(n)$ operácií;

krok 3: $O(\log n)$ času, $O(n)$ operácií;

krok 4: Všetky vrcholy s rovnakou farbou v následných pamäťových miestach; poznáme miesta prvého a posledného vrchola. Prefarbenie potrvá $O(1)$ paralelného času a $O(n_i)$ operácií (n_i je počet vrcholov farby i).

□

2.7.4 Superrýchly 3-farbiaci algoritmus

Nech $t > 3$ je počet bitov preprezentujúci farby v počiatočnom farbení c. Potom c' môže byť reprezentované s $\lceil \log t \rceil + 1$ bitmi. Teda ak q je počet farieb v c, potom c' používa $O(\log q)$ farieb. Procedúra 2.7.1 môže byť opakovane použitá (pokiaľ $t > \lceil \log t \rceil + 1$ také, že $t > 3$). Algoritmus 2.7.1 pre $t = 3$ dá farbenie s 6 (???) farbami. Počet iterácií 2.7.1 je $\log^* x = \min\{i \mid \log^{(i)} x \leq 1\}$. Počet farieb bude zredukovaný na menej ako 6 po $O(\log^* n)$ iteráciách. Zredukujeme počet farieb na 3:

pre každé $3 \leq \ell \leq 5$: pre každý vrchol i farby ℓ prefarbíme i s najmenšou možnou farbou z $\{0,1,2\}$ (napr. rôznou od predchodcu a nasledovníka)

Lema 2.7.3: *Vrcholy orientovanej kružnice možno zafarbiť tromi farbami v čase $O(\log^* n)$ použitím $O(n \log^* n)$ operácií.*

EREW PRAM: Simultánnemu prístupu k $c(i)$ sa možno vyhnúť vytvorením kópie c a pristupovaním ku kópii farby nasledovníka.

2.8 Resumé

Algoritmus	Čas	Práca	Metóda	PRAM
Prefixové sumy	$O(\log n)$	$O(n)$	Vyvážený bin. strom	EREW
Nerekurzívne prefix. sumy	$O(\log n)$	$O(n)$	Vyvážený bin. strom	EREW
Pointer Jumping	$O(\log h)$	$O(n \log h)$	Pointer Jumping	CREW
Paralelný prefix Zakorenené stromy	$O(\log h)$	$O(n \log h$	Pointer Jumping	CREW
Vrchný obal	$O(\log^2 n)$	$O(n \log n)$	Divide & Conquer	CREW
Rozdelenie	$O(\log n)$	$O(n+m)$	Rozkladanie	CREW
Spojenie	$O(\log n)$	$O(n)$	Rozkladanie	CREW
Základné maximum	$O(1)$	$O(n^2)$	Porovnanie $\forall$ dvojíc	Cmn. CRCW
Rýchle/Opt. maximum	$O(\log \log n)$	$O(n)$	Accel. Cascading	Cmn. CRCW
Základné farbenie	$O(1)$	$O(n)$	Lámanie symetrie	EREW
Rýchle farbenie	$O(\log^* n)$	$O(n \log^* n)$	Lámanie symetrie	EREW
3-farbenie kružnice	$O(\log n)$	$O(n)$	Lámanie symetrie Integer Sort	EREW

2.9 Cvičenia

(1) Nech $A = (a_1, ..., a_n)$ je pole prvkov. "Sufixové minimum" je minimum spomedzi $\{a_i, ..., a_n\}$ pre všetky $1 \leq i \leq n$. Vymyslite algoritmus pracujúci v čase $O(\log n)$ pre suxixové a prefixové minimum A použitím $O(n)$ operácií na EREW PRAM.

(2) Nech A je `boolean` pole veľkosti n.

(a) Vymyslite algoritmus pracujúci v čase $O(1)$ na CRCW PRAM, ktorý nájde najmenší index k taký, že $A(k) = 1$. Celkový počet operácií musí byť $O(n)$. HINT: Použite $\sqrt{n}$ Divide&Conquer stratégiu. Začnite riešením podproblémov.

(b) V akom čase možno riešiť tento problém na CREW PRAM ? Váš algoritmus musí používať $O(n)$ operácií.

(3) Nech T je 2-3 strom s n listami. Chceme vyhľadať danú množinu k prvkov v T, kde $k < n$. Ukážte, ako toto dosiahnuť v čase $O(\log n)$ na EREW PRAM.

(4) Nech $T = (V, E)$ je zakorenený strom špecifikovaný $(i, p(i))$, $1 \leq i \leq |V| = n$, kde $p(i)$ je rodič i. Ukážte, že T môže byť vždy 2-zafarbený. Vymyslite algoritmus pracujúci v čase $O(\log n)$ na CREW PRAM, ktorý nájde takéto farbenie.

(5) Je daných n nezávislých úloh s časmi vykonávania $\{t_1, ..., t_n\}$, ktoré určujú rozvrh každej úlohy na množine m strojov tak, že čas ukončenia je maximum. Úloha môže byť rozdelená na rôzne stroje. Váš algoritmus by mal bežať v čase $O(\log n)$ použitím celkovo $O(n)$ operácií.

3. Zoznamy a stromy

3.1 List-ranking

Nech L je spájaný zoznam n uzlov, špecifikovaný poľom $S(i)$. *List-ranking problém* je určiť vzdialenosť každého uzla i od konca zoznamu.

Riešenie pomocou techniky Pointer-Jumping: $T(n) = O(\log n)$, $W(n) = O(n \log n)$.

3.1.1 Jednoduchý optimálny list-ranking algoritmus

Algoritmus 3.1.1: (List-ranking s použitím Pointer-Jumping)

vstup: Spájaný zoznam n uzlov taký, že **(1)** nasledovník každého uzla i je daný $S(i)$ **(2)** S hodnota posledného uzla zoznamu je 0.

výstup: Pre každé $1 \leq i \leq n$ vzdialenosť $R(i)$ uzla i od konca zoznamu.

```
begin
 1: for 1 ≤ i ≤ n pardo
        if S(i) ≠ 0
            then set R(i) := 1
            else set R(i) := 0;
 2: for 1 ≤ i ≤ n pardo
        begin
            set Q(i) := S(i);
            while (Q(i) ≠ 0) and (Q(Q(i)) ≠ 0) do begin
              set R(i) := R(i) + R(Q(i));
              set Q(i) := Q(Q(i));
            end
        end
end.
```

Čas $O(\log n)$, počet operácií $O(n \log n)$. EREW PRAM

3.1.2 List-ranking stratégia

(1) Zmenšíme spájaný zoznam L až pokiaľ neostane $O(\frac{n}{\log n})$ uzlov.

(2) Použijeme techniku Pointer-Jumping na krátky zoznam ostávajúcich uzlov.

(3) Obnovíme pôvodný zoznam a ohodnotíme všetky vrcholy odstránené v kroku 1.

3.1.3 Nezávislé množiny

Množina $\mathcal{I}$ vrcholov je *nezávislá*, ak vždy keď $i \in \mathcal{I}$, tak $S(i) \in \mathcal{I}$.

Algoritmus 3.1.2: (Odstránenie uzlov z nezávislej množiny)

vstup: **(1)** Polia S a P dĺžky n reprezentujúce relácie nasledovníka a predchodcu spájaného zoznamu **(2)** nezávislá množina $\mathcal{I}_S$ taká, že $P(i), S(i) \neq 0$ **(3)** hodnota $R(i)$ pre každý vrchol i.

výstup: Zoznam získaný po odstránení všetkých uzlov v $\mathcal{I}$ s aktualizovanými R hodnotami.

begin

1: Priraď postupné sériové čísla $N(i)$ prvkom $\mathcal{I}$, kde $1 \leq N(i) \leq |\mathcal{I}| = n'$;

2: for $\forall i \in \mathcal{I}$ pardo

begin

set $U(N(i)) := (i, S(i), R(i))$;
set $R(P(i)) := R(P(i)) + R(i)$;
set $S(P(i)) := S(i)$;
set $P(S(i)) := P(i)$;

end

end.

Čas $O(\log n)$, počet operácií $O(n)$.

3.1.4 Určenie nezávislej množiny

Je dané k-farbenie zoznamu L. Vrchol u je *lokálne minimum* (*lokálne maximum*), ak farba u je menšia (väčšia) ako farby predchodcu aj nasledovníka uzla u.

Lema 3.1.1: *Nech je dané k-farbenie zoznamu L veľkosti n. Množina $\mathcal{I}$ lokálnych miním (maxím) je nezávislá množina veľkosti $\Omega(\frac{n}{k})$. Množina $\mathcal{I}$ môže byť identifikovaná v čase $O(1)$ použitím lineárneho počtu operácií.*

- *u, v sú susediace lokálne minimá. Maximálny počet uzlov medzi u, v je $2k - 3$.*
- *Množina lokálnych miním má veľkosť $\Omega(\frac{n}{k}) \geq \frac{n}{5}$.*
- *$\mathcal{I}$ môže byť identifikovaná v čase $O(1)$ (kontrolovaním farby predchodcu a nasledovníka).*

Algoritmus 3.1.3: (Jednoduchý optimálny List-Ranking)

vstup: Spájaný zoznam s n uzlami taký, že nasledovník každého uzla i je daný $S(i)$.

výstup: Pre každý uzol i jeho vzdialenosť od konca zoznamu.

begin

1: set $n_0 := n$;
set $k := 0$;

2: while $n_k > \frac{n}{\log n}$ do

begin

2.a: set $k := k + 1$;
2.b: Zafarbi zoznam 3 farbami a identifikuj množinu $\mathcal{I}$ lokálnych miním;
2.c: Odstráň uzly v $\mathcal{I}$ (algoritmus 3.1.2);
2.d: Nech n_k je veľkosť ostávajúceho zoznamu. Stlač tento zoznam do po sebe nasledujúcich pamäťových miest;

end

3: Použi techniku Pointer-Jumping na výsledný zoznam;

4: Obnov pôvodný zoznam a ohodnoť všetky odstránené uzly postupom z kroku 2, ale obrátene;

end.

Veta 3.1.2: *Algoritmus 3.1.3 ohodnotí spájaný zoznam L s n uzlami v čase $O(\log n \log\log n)$ použitím lineárneho počtu operácií.*

Dôkaz 3.1.2: $n_{k+1} \leq \frac{4}{5} n_k \Longrightarrow n_k \leq \left(\frac{4}{5}\right)^k n \Longrightarrow$ počet iterácií na redukovanie veľkosti zoznamu pod $\frac{n}{\log n}$ je $O(\log\log n)$.
$|\mathcal{I}| \geq \frac{n_k}{5}$;
Čas $O(\log n \log\log n)$.
$W(n) = O\left(\sum_k n_k\right) = O\left(\sum_k \left(\frac{4}{5}\right)^k n\right) = O(n)$. □

3.2 Technika Eulerovských ťahov

Nech $T = (V, E)$ je strom. $T' = (V', E')$ je orientovaný graf. (u, v) v T zameníme za $\langle u, v\rangle, \langle v, u\rangle$ v T'. T' je eulerovský graf.

Eulerovský ťah grafu $T' = (V', E')$ je definovaný funkciou nasledovníka s zobrazujúcou hrany $e \in E'$ do hrán $s(e) \in E'$. Pre každý vrchol $v \in V$ stanovíme usporiadanie susedných vrcholov: $\mathtt{adj}(v) = \langle u_0, u_1, ..., u_{d-1}\rangle$. $s(\langle u_i, v\rangle) = \langle v, u_{(i+1) \bmod d}\rangle$ pre $0 \leq i \leq d-1$.

Lema 3.2.1: *Je daný strom $T = (V, E)$ a usporiadanie množiny vrcholov susedných s každým vrcholom $v \in V$. Funkcia s definovaná vyššie určuje eulerovský ťah v $T' = (V', E')$.*

Lema 3.2.2: *Je daný strom $T = (V, E)$ definovaný zoznamom susedností jeho vrcholov s "prídavnými ukazovateľmi". Môžeme zostrojiť eulerovskú kružnicu v T' v čase $O(1)$ použitím $O(n)$ operácií, kde $n = |V|$.* EREW PRAM

3.2.1 Výpočty na stromoch

Zakorenenie stromu

Algoritmus 3.2.1: (Zakorenenie stromu)

vstup: **(1)** Strom T definovaný zoznamami susedností jeho vrcholov **(2)** eulerovský ťah definovaný funkciou nasledovníka s **(3)** špeciálny vrchol r.

výstup: Pre každý vrchol $v \neq r$ rodič $p(v)$ vrchola v v strome s koreňom r.

begin

1: Identifikuj posledný vrchol u objavujúci sa na zozname susedností r; set $s(\langle u, r\rangle) = 0$;

2: Priraď váhu 1 každej hrane $\langle x, y\rangle$ a použi "paralelný prefix" na zoznam hrán definovaný funkciou s;

3: Pre každú hranu $\langle x, y\rangle$ polož $x = p(y)$ vždy, keď $\mathtt{prefixsum}(\langle x, y\rangle) < \mathtt{prefixsum}(\langle y, x\rangle)$;

end.

Lema 3.2.3: *Je daný strom $T = (V, E)$ reprezentovaný cyklickými zoznamami susedností s prídavnými ukazovateľmi (popísanými vyššie) a špeciálny vrchol r. Môžeme zakoreniť strom T tak, že určíme rodiča $p(v)$ každého vrchola $v \in V$ v čase $O(\log n)$ a $O(n)$ operáciami, kde $n = |V|$.*

Post-orderové očíslovanie

Algoritmus 3.2.2: (Post-orderové očíslovanie)

vstup: **(1)** Zakorenený strom T s koreňom r **(2)** odpovedajúci Eulerovský ťah definovaný funkciou s.

výstup: Post-orderové číslo $\texttt{post}(v)$ každého vrchola v.

begin

1: Pre každý vrchol $v \neq r$ priraď váhy $w(\langle v, p(v)\rangle) = 1$ a $w(\langle p(v), v\rangle) = 0$;

2: Vykonaj "paralelný prefix" na zoznam hrán definovaný funkciou s;

3: Pre každý vrchol $v \neq r$ set $\texttt{post}(v) := \texttt{prefixsum}(\langle v, p(v)\rangle)$;
Pre $v = r$ set $\texttt{post}(r) = n$, kde $n = |T|$;

end.

Výpočet úrovne vrchola

- Priraď $w(\langle p(v), v\rangle) = +1$ a $w(\langle v, p(v)\rangle) = -1$
- $\texttt{level}(v) = \texttt{prefixsum}(\langle p(v), v\rangle)$

Výpočet počtu potomkov

Veta 3.2.4: *Nech T je strom s n vrcholmi, zakorenený v r, daný zoznamami susedností. Každý výpočet môže byť vykonaný optimálne v čase $O(\log n)$ na* EREW PRAM*:*

(1) *Výpočet post-orderového čísla každého vrchola.*
(2) *Výpočet úrovne každého vrchola.*
(3) *Výpočet pre-orderového čísla každého vrchola.*
(4) *Výpočet počtu potomkov každého vrchola.*

Paralelný prefix EP; $w(\langle p(v), v\rangle) = 0$, $w(\langle v, p(v)\rangle) = 1$;
$\texttt{size}(v) = \texttt{prefixsum}(\langle v, p(v)\rangle) - \texttt{prefixsum}(\langle p(v), v\rangle)$

3.3 Kontrakcia stromu

Problém vyhodnocovania výrazov.

3.3.1 Operácia *Rake*

Nech $T = (V, E)$ je zakorenený binárny strom s koreňom r taký, že $p(v)$ reprezentuje rodiča v. *Rake* použitá na u $(p(u) \neq r)$ pozostáva z odstránenia u a $p(u)$ z T a pripojenia súrodenca u označeného $\texttt{sib}(u)$ k $p(p(u))$.

Algoritmus 3.3.1: (Kontrakcia stromu)

vstup: **(1)** Zakorenený binárny strom T taký, že každý vrchol má práve dvoch potomkov **(2)** pre každý vrchol $v \neq r$ rodič $p(v)$ a súrodenec $\texttt{sib}(v)$.

výstup: T skontrahovaný na binárny strom s tromi vrcholmi.

begin

1: Postupne označíme listy v poradí zľava doprava, vyjmúc najľavejší a najpravejší list a uložíme označené listy do poľa A veľkosti n;

2: for $\lceil \log(n+1) \rceil$ iterácií do

begin

2.a: Použijeme ***rake*** operáciu súčasne na všetky prvky $A_{\text{nepárne}}$, ktoré sú ľavými potomkami;
2.b: Použijeme ***rake*** operáciu súčasne na ostatok prvkov $A_{\text{nepárne}}$;
2.c: set $A := A_{\text{párne}}$;

end

end.

$O(\log n)$ čas na EREW PRAM; // $m \xrightarrow{1 \text{ iterácia}} \lfloor \frac{m}{2} \rfloor$; m je počet listov.
$W(n) = O(\sum_i \frac{n}{2^i}) = O(n)$ // $|A_{\text{párne}}| \leq \frac{|A|}{2}$

3.3.2 Vyhodnocovanie aritmetických výrazov

S každým vrcholom $v \in V$ asociujeme návestie (a_v, b_v) reprezentujúce lineárny výraz $a_v X + b_v$. *Invariant (I)*: Nech w je vnútorný vrchol aktuálneho stromu taký, že u obsahuje operátor $\bullet \in \{+, \times\}$ a má potomkov v, w, ktorých návestia sú príslušne (a_v, b_v) a (a_w, b_w). Potom hodnota podvýrazu u je $\texttt{val}(u) = (a_v \texttt{val}(v) + b_v) \bullet (a_w \texttt{val}(w) + bw)$.

Veta 3.3.1: *Je daný binárny strom T reprezentujúci aritmetický výraz. Algoritmus 3.3.1 so zväčšenou* rake *operáciou môže byť použitý na výpočet* $\texttt{val}(T)$ *v čase* $O(\log n)$ *použitím lineárneho počtu operácií.*

Zväčšená rake operácia: Hodnota u je daná podľa $\texttt{val}(u) = (a_v c_v + b_v) \bullet_u (a_w X + b_w)$; ($X$ je neznáma hodnota (???)) Príspevok $\texttt{val}(u)$ k uzlu $p(u)$ je daný podľa

$$E = a_u \texttt{val}(u) + b_u = a_u[(a_v c_v + b_v) \bullet_u (a_w X + b_w)] + b_u$$

3.3.3 Výpočet stromových funkcií

Nech $(U, *)$ je komutatívny monoid[1] s jednotkou e. $T = (V, E)$ je strom, v ktorom každý vrchol je označený návestím $L(v) \in U$.

Problém: Pre každý v vypočítať $L(v) \in U$ ktoré je výsledkom použitia $*$ na všetky prvky v zakorenenom podstrome na v.

Cv.(?): Nech $*$ je operátor minima. Potom $L(v)$ je minimálny prvok v podstrome zakorenenom vo v.

Veta 3.3.2: *Je daný ľubovoľný strom $T = (V, E)$ taký, že každý vrchol je označený s prvkom z lineárne usporiadanej množiny. Potom môžeme vypočítať minimálne návestie v strome zakorenenom vo v pre každý $v \in V$ v čase $O(\log n)$ použijúc lineárny počet operácií.*

- Z ľubovoľného stromu skonštruujeme binárny strom:
- S každým novým vnútorným vrcholom asociujeme návestie e (jednotku monoidu).
- Vyhodnotíme všetky podvýrazy asociované so všetkými tými vrcholmi (použitím algoritmu 3.3.1 s *rake.* item Čas $O(\log n)$, práca $O(n)$.)

[1] pologrupa s jednotkou

3.4 Najnižší spoloční predkovia

Najnižší spoločný predok (LCA[2]) dvoch vrcholov u, v ($\texttt{lca}(u, v)$) je vrchol w taký, že predok u, v je najďalej od koreňa.

LCA problém: Špeciálne prípady: **cesta** a **kompletný binárny strom**.

Cesta: Vypočítať vzdialenosť všetkých vrcholov od koreňa. Odpoveď na ľubovoľnú otázku $lca(u, v)$ získame v čase $O(1)$ vypočítaním vzdialeností u, v od koreňa.

Kompletný binárny strom: Predpočítať *inorderové číslo* každého vrchola stromu T. Potom $\texttt{lca}(x, y)$ môže byť vypočítané: vyjadriť x, y ako binárne čísla, nech i je prvá pozícia zľava taká, že x, y nesúhlasia. $\texttt{lca}(x, y)$ je rovné číslu vyjadrenému binárnym zápisom

$$z_1 z_2 \dots z_{i-1} 10 \dots 0,$$

kde $z_1 \dots z_{i-1}$ je najľavejších $i - 1$ bitov, na ktorých sú x, y identické.

Príklad 3.4.1: $9 = (1001)_2$, $13 = (1101)_2$ ... teda $\texttt{lca}(9, 13) = (1100)_2 = 12$ ♠

Riešenie LCA problému: opatrne zobraziť T do binárneho stromu s logaritmickou hĺbkou.

3.4.1 Redukcia na problém minima intervalu

Vstupný strom T. Skonštruujeme eluerovský ťah v T. Vymeníme každé $\langle u, v \rangle$ za v. Definujeme *Eulerovské pole* A ako odpovedajúce usporiadané vrcholy s koreňom na začiatku. Ak $n = |V|$, potom veľkosť $|A| = 2n - 1$. Pole B je úroveň každého prvku A.

Príklad 3.4.2: $A = (1, 2, 3, 2, 4, 5, 4, 6, 4, 7, 4, 2, 1, 8, 1, 9, 1)$
$B = (0, 1, 2, 1, 2, 3, 2, 3, 2, 3, 2, 1, 0, 1, 0, 1, 0)$ ♠

$\ell(v)$ a $r(v)$ sú indexy najľavejšieho a najpravejšieho výskytu v v A. (**pozn.:** prvok $a_i = v$ je najľavejší výskyt v v $A \iff \texttt{level}(a_{i-1}) = \texttt{level}(v) - 1$; prvok $a_i = v$ je najpravejší výskyt v v $A \iff \texttt{level}(a_{i+1}) = \texttt{level}(v) + 1$.) $\ell(v)$, $r(v)$ môžu byť vypočítané v čase $O(1)$ použijúc linárny počet operácií.

Lema 3.4.1: *Je daný zakorenený strom $T = (V, E)$. Nech A, $\texttt{level}(v)$, $\ell(c)$, $r(v)$ (pre $v \in V$) sú ako definované vyššie. Nech $u, v \in V$. Potom platia nasledovné tvrdenia:*

(1) *u je predok v práve vtedy, keď $\ell(u) < \ell(v) < r(u)$;*
(2) *u, v nie sú príbuzní; t.j. u nie je potomok v a v nie je potomok u práve vtedy, keď buď $r(u) < \ell(v)$ alebo $r(v) < \ell(u)$;*
(3) *Ak $r(u) < \ell(v)$, potom $\texttt{lca}(u, v)$ je vrchol s minimálnou úrovňou nad intervalom $[r(u), \ell(v)]$*

Dôkaz 3.4.1:

(1) u je predok $v \implies u$ je navštívený pred $v \implies$ podstrom zakorenený vo v je úplne prehľadaný pred posledným výskytom v v DFS (t.j. eulerovskom ťahu) $\implies \ell(u) < \ell(v) < r(u)$.

$\ell(u) < \ell(v) < r(u)$ a u nie je predkom $v \implies$ podstrom zakorenený v u je kompletne spracovaný pred prvou návštevou $v \implies r(u) < \ell(v)$, čo je spor.

(3) $r(u) < \ell(v) \implies$ vrcholy s úrovňou v $[r(u), \ell(v)]$ sú na ceste $u \to v$ alebo ich nasledovníkov (???). Takže vrchol s minimálnou úrovňou je $\texttt{lca}(u, v)$.

□

[2]from English: Lowest Common Ancestor

3.4.2 Problém minima intervalu

$B = \texttt{level}(A)$. Je daný ľubovoľný interval $[k, j]$, kde $1 \leq k \leq j \leq n$. Nájdime *minimum* $\{b_k, ..., b_j\}$.

Základný problém minima

Prefixové minimum z $B = (b_1, ..., b_n)$ sú prvky poľa $(c_1, ..., c_n)$ také, že $c_i = \min\{b_1, ..., b_i\}$ pre $1 \leq i \leq n$. Podobne definujeme *sufixové minimum* z B. Prefixové a sufixové minimá môžu byť vypočítané v čase $O(\log n)$ a práci $O(n)$ pomocou algoritmu pre prefixové súčty.

Algoritmus 3.4.1: (Základné minimum intervalu)

vstup: Pole B veľkosti $n = 2^\ell$, kde ℓ je kladné celé číslo.

výstup: Úplný binárny strom s pomocnými premennými $P(h, j)$ a $S(h, j)$, $0 \leq h \leq \log n$, $1 \leq j \leq \frac{n}{2^h}$ takými, že $P(h, j)$ a $S(h, j)$ reprezentujú prefixové a sufixové minimá podpoľa definovaného listami podstromu zakoreneného v (h, j).

```
begin
1: for 1 ≤ j ≤ n pardo
       begin
           set P(0, j) := B(j);
           set S(0, j) := B(j);
       end
2: for h = 1 to log n do
       for 1 ≤ j ≤ n/2^h pardo
           begin
             Spoj P(h − 1, 2j − 1) a P(h − 1, 2j) do P(h, j);
             Spoj S(h − 1, 2j − 1) a S(h − 1, 2j) do S(h, j);
           end
end.
```

Čas $O(\log n)$ použitím $O(n \log n)$ operácií.

Príklad 3.4.3: $B = (5, 10, 3, 4, 7, 1, 8, 2)$ $P(2, 1) = (5, 5, 3, 3)$ $P(2, 2) = (7, 1, 1, 1)$ $S(2, 1) = (4, 3, 3, 3)$ $S(2, 2) = (2, 2, 1, 1)$ $P(3, 1) = (5, 5, 3, 3, 3, 1, 1, 1)$ $S(3, 1) = (2, 2, 1, 1, 1, 1, 1, 1)$ ♠

Je daná požiadavka (query) minima z intervalu $[i, j]$. Nech $v = \texttt{lca}(i, j)$. Vrchol v možno nájsť v sekvenčnom čase $O(1)$. Nech u, w sú ľavý a pravý potomok vrchola v. Potom $\min\{a_i, ..., a_j\}$ je minimom dvoch prvkov: sufixového minima prislúchajúceho prvku i v S poli w a prefixového minima prislúchajúceho prvku j v P poli w.

Príklad 3.4.4: $i = 2$, $j = 5$ a teda $B(2) = 10$ a $B(5) = 7$. $\texttt{lca}(2, 5)$ je koreň $(3, 1)$ s potomkami $(2, 1)$ a $(2, 2)$. $S(2, 1) = (4, 3, 3, 3)$ a $P(2, 2) = (7, 1, 1, 1)$. Teda $\min\{3, 7\} = 3$. ♠

Lema 3.4.2: *Algoritmus 3.4.1 spracuje vstupné pole B v čase $O(\log n)$ použitím celkovo $O(n \log n)$ operácií. Každá otázka minimálneho intervalu (minimum-range) môže byť zodpovedaná v sekvenčnom čase $O(1)$.* (CREW PRAM)

Modifikácia pre EREW PRAM bez asymptotického zhoršenia v hraniciach zložitosti.[3]

[3] pozn.prekl.: ???

3.4.3 Optimálny základný algoritmus minima intervalu

Predspracovanie: EREW PRAM

(1) Rozdeľ B do blokov B_i rovnakej veľkosti $\log n$.

(2) Predspracuj každý blok B_i osobitne pre problém minima intervalu použitím optimálneho sekvenčného algoritmu. Teda otázka, ktoré dva prvky patria do toho istého bloku B_i môže byť spracovaná v sekvenčnom čase $O(1)$.

(3) Pre každé B_i vypočítaj minimálny prvok x_i a jeho prefixové a sufixové minimum.

(4) Predspracuj pole $(x_1, ..., x_{\frac{n}{\log n}})$ ako v algoritme 3.4.1.

Teraz môžeme zodpovedať otázku minima intervalu v konštantnom čase:

Predpokladajme, že máme danú otázku na minimum intervalu nad intervalom $[i, j]$, $1 \leq i \leq j \leq n$. Nech i', j' sú indexy blokov obsahujúcich a_i, a_j. Uvažujme tieto prípady:

$i' = j'$: Otázka minima intervalu je redukovaná na otázku na jedinom $B_{i'}$. Čas $O(1)$

$j' = i' + 1$: Odpoveď je minimum dvoch prvkov: sufixového minima v $B_{i'}$ prislúchajúceho k a_i a prefixového minima v $B_{j'}$ prislúchajúceho k a_j. Čas $O(1)$.

$j' > i' + 1$: Použijeme binárny strom na minimálnych prvkoch x_i (krok 4). Zodpovieme otázku prislúchajúcu k $[i' + 1, j' - 1]$ ako v algoritme 3.4.1. Odpoveď na otázku je minimum troch prvkov: prvku získaného použitím binárneho stromu (alg.3.4.1), sufixového minima prislúchajúceho k a_i v $B_{i'}$ a prefixového minima prislúchajúceho k a_j v $B_{j'}$. Čas $O(1)$

3.5 Resumé

Algoritmus	Čas	Práca	PRAM
3.1.1 List-ranking s použitím Pointer Jumping	$O(\log n)$	$O(n \log n)$	EREW
3.1.2 Odstránenie uzlov z nezávislej množiny	$O(\log n)$	$O(n)$	EREW
3.1.3 Jednoduchý optimálny list-ranking	$O(\log n \log\log n)$	$O(n)$	EREW
Kontrakcia zoznamu[4] (jedna úroveň)	$O(1)$	$O(\frac{n}{\log n})$	EREW
List-ranking	$O(\log n)$	$O(n)$	EREW
Konštrukcia Eulerovského ťahu	$O(1)$	$O(n)$	EREW
3.2.1 Zakorenenie stromu	$O(\log n)$	$O(n)$	EREW
3.2.2 Post-orderové číslovanie Výpočet úrovne vrchola Výpočet počtu potomkov	$O(\log n)$	$O(n)$	EREW
3.3.1 Kontrakcia stromu	$O(\log n)$	$O(n)$	EREW
Vyhodnocovanie aritmetických výrazov	$O(\log n)$	$O(n)$	EREW
Výpočet stromových funkcií	$O(\log n)$	$O(n)$	EREW
3.4.1 Základné minimum intervalu	$O(\log n)$	$O(n \log n)$	CREW
Minimum intervalu	$O(\log n)$	$O(n)$	CREW
LCA	$O(\log n)$	$O(n)$	CREW

3.6 Cvičenia

(1) Prepokladajme, že použijeme $O(\log^* n)$-časový algoritmus pre 3-farbenie na identifikáciu veľkej nezávislej množiny potrebnej v algoritme 3.1.3. Viete prinútiť výsledný list-ranking algoritmus bežať v čase $O(\log n)$ s $O(n \log^* n)$ operáciami ? Zdôvodnite svoju odpoveď !

(2) $T = (V, E)$ je zakorenený strom. Ukážte, ako získať preorderové číslovanie každého vrchola v optimálne v čase $O(\log n)$ na EREW PRAM modeli, kde $n = |V|$.

(3) Nech $T = (V, E)$ je ľubovoľný strom s $|V| = n$. Vrchol v je ťažisko T, ak odstránenie v generuje podstromy každý veľkosti menšej alebo rovnej $\frac{n}{2}$. Vyrobte optimálny $O(\log n)$-časový EREW PRAM algoritmus na nájdenie ťažiska T.

(4) (a) Navrhnite $O(1)$-časový algoritmus pre výpočet prefixového a sufixového minima n-prvkového poľa A použijúc celkovo $O(n^2)$ operácií.

(b) Použite $\sqrt{n}$ DnC stratégiu na získanie $O(\log \log n)$-časového algoritmu. Celkový počet použitých operácií musí byť $O(n)$. Špecifikujte použitý PRAM model.

(5) Použitím $O(\log \log n)$-časového algoritmu na výpočet prefixového minima poľa veľkosti n (cvičenie **3.(4)(b)**) vyrobte algoritmus na riešenie problému minima intervalu v čase $O(\log \log n)$. Aký je celkový počet použitých operácií ? Viete spraviť váš algoritmus optimálnym ? Vysvetlite svoju odpoveď ! Špecifikujte použitý PRAM model !

4. Vyhľadávanie, spájanie a triedenie

4.1 Vyhľadávanie

$X = (x_1, ..., x_n)$ je n rôznych prvkov z lineárne usporiadanej množiny $(S, \leq)$ takých, že $x_1 < x_2 < ... < x_n$. Je dané $y \in S$. **Problém vyhľadávania** je určiť i také, že $x_i \leq y \leq x_{i+1}$ (kde $x_0 = -\infty$, $x_{n+1} = +\infty$).

Algoritmus 4.1.1: (Paralelné vyhľadávanie pre procesor P_j)

vstup: **(1)** Pole $X = (x_1, ..., x_n)$ také, že $x_1 < ... < x_n$ **(2)** prvok y **(3)** počet p procesorov, kde $p \leq n$ **(4)** číslo procesora j, kde $1 \leq j \leq p$.

výstup: Index i taký, že $x_i \leq y < x_{i+1}$.

begin

1: if $j = 1$ then begin

set $\ell := 0$; set $r := n + 1$; set $x_0 := -\infty$; set $x_{n+1} := +\infty$; set $c_0 := 0$; set $c_{p+1} := 1$;

end;

2: while $r - \ell > p$ do begin

2.a: if $j = 1$ then begin

set $q_0 := \ell$; set $q_{p+1} := r$;

end;

2.b: set $q_j := \ell + j\frac{r-\ell}{p+1}$;

2.c: if $y = x_{q_j}$ then begin

return(q_j); exit;

end else

set $c_j := 0$ if $y > x_{q_j}$ and $c_j := 1$ if $y < x_{q_j}$;

2.d: if $c_j < c_{j+1}$ then begin

set $\ell := q_j$; set $r := q_{j+1}$;

end;

2.e: if $j = 1 \wedge c_0 < c_1$ then begin

set $\ell := q_0$; set $r := q_1$;

end;

end;

3: if $j < r - \ell$ then begin

3.a: if $y = x_{\ell+j}$ then begin

return$(\ell + j)$; exit;

end else

set $c_j := 0$ if $y > x_{\ell+j}$ and set $c_j := 1$ if $y < x_{\ell+j}$;

```
        3.b: if c_{j-1} < c_j then return(ℓ + j − 1);
    end;
end.
```

Príklad 4.1.1: $X = (2, 4, 6, ..., 30)$ a $y = 19$. Predpokladajme, že $p = 2$.

`while` slučka:

iterácia	1	2	3
q_0	0	5	7
q_1	5	6	8
q_2	10	7	9
q_3	16	10	10
c_0	0	0	0
c_1	0	0	0
c_2	1	0	0
c_3	1	1	1
ℓ	5	7	9
r	10	10	10

♠

Veta 4.1.1: *Je dané pole $X = (x_1, ..., x_n)$ s $x_1 < x_2 < ... < x_n$ a prvok y. Algoritmus 4.1.1 určí index i taký, že $x_i \leq y < x_{i+1}$. Potrebný paralelný čas je*

$$O\left(\frac{\log(n+1)}{\log(p+1)}\right),$$

kde p je číslo použitého procesora.

Dôkaz 4.1.1: $s_0 := n + 1$; $s_i \leq \frac{n+1}{(p+1)^i} + p + 1 \Longrightarrow$ počet operácií je

$$O\left(\frac{\log(n+1)}{\log(p+1)^i}\right)$$

Každá operácia trvá čas $O(1)$. p-procesorový CREW PRAM

□

4.2 Spájanie

Spájací algoritmus založený na stretégii rozkladania v čase $O(\log n)$ s prácou $O(n)$. Teraz $O(\log \log n)$-časové spájanie na CREW PRAM. (**pozn.:** výpočet maxima vyžaduje čas $\Omega(\log n)$ na CREW PRAM)

4.2.1 Ohodnotenie krátkej postupnosti v utriedenej postupnosti

X je utriedená postupnosť s n rôznymi prvkami; V je postupnosť veľkosti $m = O(n^s)$, $0 < s < 1$. Algoritmus 4.1.1 môže byť použitý na ohodnotenie každého prvku V v X osobitne. Nech $p = \lfloor \frac{n}{m} \rfloor = \Omega(n^{1-s})$. Potom každý prvok V môže byť ohodnotený v X v čase $O\left(\frac{\log(n+1)}{\log(p+1)}\right) = O(1)$. Celkový počet operácií použitých na ohodnotenie každého takého prvku je $O(\frac{n}{m})$, pretože $p = \lfloor \frac{n}{m} \rfloor$.

Lema 4.2.1: *Nech V je ľubovoľná postupnosť s m prvkami a nech X je utriedená postupnosť s n rôznymi prvkami taká, že $m = O(n^s)$ pre nejakú konštantu $0 < s < 1$. Potom všetky prvky V môžu byť ohodnotené v X v čase $O(1)$ použijúc celkovo $O(n)$ operácií.*

4.3 Rýchlý spájací algoritmus

Algoritmus 4.3.1: (Ohodnotenie utriedenej postupnosti v ďalšej utriedenej postupnosti)

vstup: Dve polia $A = (a_1, ..., a_n)$ a $B = (b_1, ..., b_n)$ vo vzostupnom usporiadaní. Predpokladajme, že $\sqrt{m}$ je celé; inak zameníme $\sqrt{m}$ vždy, keď sa vyskytne, za $\lfloor\sqrt{m}\rfloor$.

výstup: Pole $\texttt{rank}(A : B)$.

begin

1: Ak $m < 4$, potom ohodnotíme prvky B použitím algoritmu 4.1.1 s $p = n$;
exit;

2: Súčasne ohodnotíme prvky $b_{\sqrt{m}}, b_{2\sqrt{m}}, ..., b_{i\sqrt{m}}, ..., b_m$ v A použitím algoritmu 4.1.1 s $p = \sqrt{m}$. Nech $\texttt{rank}(b_{i\sqrt{m}} : A) = j(i)$ pre $1 \leq i \leq \sqrt{m}$;
set $j(0) := 0$;

3: Pre $0 \leq i \leq \sqrt{m} - 1$ nech $B_i = (b_{i\sqrt{m}+1}, ..., b_{(i+1)\sqrt{m}-1})$ a nech $A_i = (a_{j(i)+1}, ..., a_{j(i+1)})$;
if $j(i) = j(i+1)$ then

set $\texttt{rank}(B_i : A_i) = (0, ..., 0)$

else

Rekurzívne počítaj $\texttt{rank}(B_i : A_i)$;

4: Nech $1 \leq k \leq m$ je ľubovoľný index, ktorý nie je násobkom $\sqrt{m}$ a nech $i = \left\lfloor \frac{k}{\sqrt{m}} \right\rfloor$. Potom $\texttt{rank}(b_k : A) = j(i) + \texttt{rank}(b_k : A)$.

end.

B	$b_1, \ldots, b_{\sqrt{m}-1}, b_{\sqrt{m}}, b_{\sqrt{m}+1}, \ldots, b_{2\sqrt{m}-1}, b_{2\sqrt{m}}, \ldots, b_{i\sqrt{m}+1}, \ldots\ldots, b_{(i+1)\sqrt{m}-1}, b_{(i+1)\sqrt{m}}, \ \ldots$
	$\downarrow \qquad\qquad\qquad\qquad \downarrow$
A	$a_1, \ldots\ldots\ldots, a_{j(1)}, a_{j(1)+1}, \ldots\ldots\ldots\ldots, a_{j(2)}, \ldots\ldots\ldots\ldots, a_{j(i)+1}, \ldots\ldots\ldots\ldots, \ a_{j(i+1)} \ , \ \ldots$

4.4 Optimálne rýchly spájací algoritmus

Predpokladajme $m = n$. Rozdeľíme A a B do blokov veľkosti aspoň $\lceil \log\log n \rceil$, napr. $A = (A_1, A_2, ...)$ a $B = (B_1, B_2, ...)$. Nech $A' = (p_1, p_2, ...)$, kde p_i je prvý prvok A_i. Nech $B' = (q_1, q_2, ...)$, kde q_i je prvý prvok B_i.

$$|A'| = O\left(\frac{n}{\log\log n}\right) \qquad |B'| = O\left(\frac{n}{\log\log n}\right)$$

(1) Spoj $A' = (p_1, p_2, ...)$ a $B' = (q_1, q_2, ...)$ použitím algoritmu 4.3.1.

Vysvetlenie: Algoritmus 4.3.1 vykonáva spájanie A', B' v čase $O(\log\log n)$ použijúc celkovo $O(n)$ operácií. Na tomto mieste sme vypočítali $\texttt{rank}(A' : B')$ a $\texttt{rank}(B' : A')$.

(2) Urči $\texttt{rank}(A' : B)$ a $\texttt{rank}(B' : A)$.

Vysvetlenie: Nech $\texttt{rank}(p_i : B') = r_i$. Teda p_i pasuje do B_{r_i}, pretože $q_{r_i} < p_i < q_{r_i+1}$. Pre $\forall p_i$ urči jeho umiestnenie v B_{r_i} sekvenčným algoritmom (napr. binárnym vyhľadávaním). Potrvá to $O(\log\log n)$ času za každý prvok. Pretože existuje $\leq \left\lceil \frac{n}{\log\log n} \right\rceil$ prvkov, $\texttt{rank}(A' : B)$ možno určiť v čase $O(\log\log n)$ použitím $O(n)$ operácií. Podobne určíme $\texttt{rank}(B' : A)$.

(3) Pre $\forall i$ urči $\texttt{rank}(A_i - \{p_i\} : B)$ a pre $\forall k$ urči $\texttt{rank}(B_k - \{q_k\} : A)$.

Vysvetlenie: Nech $\texttt{rank}(p_i : B) = j(i)$ a $\texttt{rank}(q_k : A) = \tilde{j}(k)$. p_i, q_k sú prvé prvky A_i, B_k a $|A_i|, |B_k|$ sú veľkosti $\leq \lceil \log\log n \rceil$. V kroku (3) určíme ranky ostávajúcich prvkov v každom A_i a B_k.

Lema 4.4.1: *Nech A a B sú utriedené postupnosti také, že $|A| = n$, $|B| = m$. Potom algoritmus 4.3.1 vypočíta pole $\texttt{rank}(B : A)$ v čase $O(\log\log n)$ použijúc $O((n+m)\log\log m)$ operácií.*

Dôkaz 4.4.1: Dokážeme správnosť ... indukciou na m.

(báza) $m = 3$: ohodnotenie (b_1, b_2, b_3) v A (krok (1)).

indukčný predpoklad pre všetky $m' < m$: ukážeme, že všetky prvky v B_i sú presne medzi $a_{j(i)}$ a $a_{j(i+1)+1}$.

$\forall p \in B_i : b_{i\sqrt{m}} < p < b_{(i+1)\sqrt{m}}$. Pretože $j(i) = \texttt{rank}(b_{i\sqrt{m}} : A), j(i+1) = \texttt{rank}(b_{(i+1)\sqrt{m}} : A)$, dostávame $a_{j(i)} < b_{i\sqrt{m}}, b_{(i+1)\sqrt{m}} < a_{j(i+1)+1}$. Teda $a_{j(i)} < p < a_{j(i+1)+1}$. Preto $\texttt{rank}(p : A) = j(i) + \texttt{rank}(p : A_i)$. A teda dôkazu správnosti vyplýva z indukcie. □

Dôkaz 4.4.1: Dokážeme zložitosť ...

krok (2):

čas: $\sqrt{m}$ volaní algoritmu 4.3.1 pre $p = \sqrt{n}$. Čas je $O\left(\frac{\log(n+1)}{\log(\sqrt{n}+1)}\right) = O(1)$.

práca: $O(\sqrt{m}\sqrt{n}) = \{$ pretože $2\sqrt{m}\sqrt{n} \leq n + m\} = O(n+m)$.

krok (3)+(4):

čas: Okrem rekurzívnych volaní budú volania trvať $O(1)$ času a $O(n+m)$ operácií. Nech $|A_i| = n_i$. Rekurzívne volania pre (B_i, A_i) potrvajú $T(n_i, \sqrt{m})$ času. $T(n,m) = \max_i T(n_i, \sqrt{m}) + O(1)$; $T(n,3) = O(1) \Longrightarrow T(n,m) = O(\log\log m)$.

práca: Pre každé rekurzívne volanie je $O(n+m) \Longrightarrow W(n,m) = O((n+m)\log\log m)$.

□

Dôsledok 4.4.2: *Nech A, B sú utriedené postupnosti dĺžky n. Spojenie A a B môže byť vykonané v čase $O(\log\log n)$ použijúc celkovo $O(n\log\log n)$ operácií.*

Lema 4.4.3: *Problém spojenia dvoch utriedených postupností dĺžky n možno vyriešiť v čase $O(\log\log n)$ použitím $O(n)$ operácií.* CREW PRAM

Správnosť: Všetky prvky A_i musia ležať medzi $b_{j(i)}$ a $b_{j(i+1)+1}$.

Predpokladajme $j(i+1) > j(i)$. Ak počet prvkov $b_{j(i)}, ..., b_{j(i+1)}$ je menší ako $\log\log n$, potom spojíme A_i a B_{r_i} v sekvenčnom čase $O(\log\log n)$. Inak existujú $q_k, ..., q_{k+s}$, ktoré ležia medzi $b_{j(i)+1}$ a $b_{j(i+1)}$. Pretože ranky $\tilde{j}(k), ..., \tilde{j}(k+s)$ prvkov $q_k, ..., q_{k+s}$ sú jasne známe, náš problém sa zredukuje na spojenie nie viac ako $s+2$ dvojíc podpostupností veľkosti najviac $O(\log\log n)$. Podpostupnosti sú disjunktné.

V skratke ... optimálny spájací algoritmus:

(1) Rozdeľ A, B na $A = (A_1, A_2, ...)$ a $B = (B_1, B_2, ...)$ také, že veľkosť každého bloku je $\leq \lceil \log\log n \rceil$. Použi algoritmus 4.3.1 na spojenie $A' = (p_1, p_2, ...)$ a $B' = (q_1, q_2, ...)$.

(2) Pre $\forall p$ umiestni p_i v B_{r_i}, kde $|B_{r_i}| \leq \lceil \log\log n \rceil$. ranky prvku p_i sú určené súčasne. Podobne pre $\forall q_i$ umiestni q_i v A.

(3) Problém spájania je redukovaný na množinu neprekrývajúcich sa spájacích podproblémov; každá z odpovedajúcich dvojíc postupností zahŕňa (involves) $O(\log\log n)$ prvkov. Všetky spájacie podproblémy môžu byť vyriešené súčasne v čase $O(\log\log n)$ použijúc lineárny počet operácií.

4.5 Triedenie

4.5.1 Jednoduchý optimálny triediaci algoritmus

Dvojcestný merge-sort algoritmus:

(1) rozdeľ vstupnú postupnosť X na X_1 a X_2 rovnakej veľkosti;

(2) utrieď X_2 a X_2 oddelene;

(3) spoj dve utriedené postupnosti;

Náš problém možno preformulovať nasledovne: Pre každý vrchol v vyváženého binárneho stromu T vypočítaj utriedený zoznam $L[v]$ obsahujúci všetky prvky uložené v podstrome zakorenenom vo v. Samozrejme, koreň bude obsahovať utriedený zoznam.

Tento proces možno dosiahnuť modifikovaným algoritmom 3.4.1, ktorý pre každý vrchol v počíta prefixové minimum prvkov vysktujúcich sa v podstrome zakorenenom vo v. Jediný rozdiel leží v spájacej procedúre (použijeme optimálny $O(\log\log n)$ spájací algoritmus).

Algoritmus 4.5.1: (Jednoduchý Merge-Sort)

vstup: Pole X rádu n, kde $n = 2^k$ pre nejaké celé ℓ.

výstup: Vyvážený binárny strom s n listami taký, že pre každé $0 \leq h \leq \log n$, $L(h,j)$ obsahuje utriedenú podpostupnosť pozostávajúca z prvkov uložených v podstrome zakorenenom v (h,j) pre $1 \leq j \leq \frac{n}{2^k}$. T.j. vrchol (h,j) obsahuje utriedený zoznam prvkov $X(2^h(j-1)+1), X(2^h(j-1)+2), ..., X(2^h j)$.

begin

1: for $1 \leq j \leq n$ pardo

$L(0,j) := X(j)$;

2: for $h = 1$ to $\log n$ do

for $1 \leq j \leq \frac{n}{2^k}$ pardo

Spoj $L(h-1, 2j-1)$ a $L(h-1, 2j)$ do utriedeného zoznamu $L(h,j)$;

end.

Veta 4.5.1: *Pre každý vrchol v vyváženého stromu T generuje algoritmus 4.5.1 utriedený zoznam $L[v]$ pozostávajúci z prvkov uložených v podstrome zakorenenom vo vrchole v. Čas behu je $O(\log n \log\log n)$ a celkový počet operácií je $O(n \log n)$. Teda algoritmus 4.5.1 je optimálny.*

Tento algoritmus vyžaduje CREW PRAM.

Optimálny $O(\log n)$-časový triediaci algoritmus je založený na pipelined merge-sort algoritme.

4.6 Triedenie sietí

Sieť komparátorov pozostáva z komparátorov (modulov so vstupmi x, y a výstupmi $\min\{x,y\}, \max\{x,y\}$. *Veľkosť* siete komparátorov je počet komparátorov v sieti. *Hĺbka* je maximálny počet komparátorov stretnutých na ceste od vstupu k výstupu.

Nevšímavé (oblivious) algoritmy

Nevšímavý porovnávací algoritmus – tok riadenia nezávisí na čiastočných hodnotách prvkov a_i, kde $(a_0, ..., a_{n-1})$ je vstup.

Nevšímavé triediace algoritmy (sa?) priamo zobrazujú na siete komparátorov.

Bitonické postupnosti

$X = (x_0, ..., x_{n-1})$ je postupnosť prvkov vytiahnutých z lineárne usporiadanej množiny, $n = 2^k$. X je *bitonická*, ak pre $j < n$ platí $x_{j \bmod n} \leq x_{(j+1) \bmod n} \leq ... \leq x_{\ell \bmod n}$ a $x_{(\ell+1) \bmod n} \geq ... \geq x_{(j+n-1) \bmod n}$ pre nejaké ℓ.

Jedinečná cross-over vlastnosť bitonických postupností

Nech $X = (x_0, ..., x_{n-1})$ je taká, že $x_0 \leq x_1 \leq ... \leq x_{\frac{n}{2}-1}$ a $x_{\frac{n}{2}} \geq x_{\frac{n}{2}+1} \geq ... \geq x_{n-1}$.

Lema 4.6.1: *Ľubovoľná bitnocká postupnosť spĺňa jedinečnú cross-over vlastnosť.*

Triedenie bitonických postupností

$X = (x_0, x_1, ..., x_{n+1})$ je bitonická postupnosť.
$L(X) = (\min\{x_0, x_{\frac{n}{2}}\}, \min\{x_1, x_{\frac{n}{2}+1}\}, ..., \min\{x_{\frac{n}{2}-1}, x_{n-1}\})$
$R(X) = (\max\{x_0, x_{\frac{n}{2}}\}, \max\{x_1, x_{\frac{n}{2}+1}\}, ..., \max\{x_{\frac{n}{2}-1}, x_{n-1}\})$

Lema 4.6.2: *Nech X je bitonická postupnosť. Potom aj $L(X)$ aj $R(X)$ sú bitonické a každý prvok $L(X)$ je menší ako každý prvok $R(X)$.*

Dôkaz 4.6.2: Podľa jedinečnej cross-over vlastnosti existuje rez medzi x_i, x_{i+1} (a $x_{\frac{n}{2}+i}, x_{\frac{n}{2}+i+1}$), ktorý indukuje dve podoblasti $\leq$ a $>$. Povedzme, že $\leq$ je navrchu.

Samozrejme, $L(X) = (x_0, ..., x_i, x_{\frac{n}{2}+i+1}, ..., x_{n-1})$ a $R(X) = (x_{\frac{n}{2}}, ..., x_{\frac{n}{2}+i}, x_{i+1}, ..., x_{\frac{n}{2}-1})$.

$L(X) \leq R(X)$. $L(X)$ a $R(X)$ sú bitonické, keďže sú podpostupnosťami bitonickej X. □

Algoritmus 4.6.1: (Bitonické spájanie)

vstup: Bitonická postupnosť $X = (x_0, ..., x_{n-1})$ taká, že $n = 2^k$, pre nejaké celé k.

výstup: Postupnosť X v utriedenom usporiadaní.

```
begin
```

1: for $0 \leq i \leq \frac{n}{2} - 1$ pardo

```
    begin
        set ℓ_i := min(x_i, x_{i+n/2});
        set r_i := max(x_i, x_{i+n/2});
    end
```

2: Použi algoritmus 4.6.1 rekurzívne na $L(X) = (\ell_0, ..., \ell_{\frac{n}{2}-1})$ a $R(X) = (r_0, ..., r_{\frac{n}{2}-1})$;

3: return(utriedená $L(X)$ nasledovaná utriedenou $R(X)$);

end.

Bitonické triediace siete (Bitonic Sorting Networks)

Veta 4.6.3: *Bitonická triediaca sieť $B(n)$ správne utriedi bitonickú postupnosť dĺžky $n = 2^k$, kde k je kladné celé. Hĺbka siete je $D(n) = \log n$ a počet komparátorov je $C(n) = \frac{1}{2}n \log n$.*

Dôkaz 4.6.3:

$$\begin{array}{ll} D(n)=1 + D\left(\frac{n}{2}\right) & C(n)=\frac{n}{2} + 2C\left(\frac{n}{2}\right) \\ D(n)=1 & C(2)=1 \end{array}$$

□

Veta 4.6.4: *Vieme zostrojiť sieť na utriedenie ľubovoľnej postupnosti dĺžky n v hĺbke $O(\log^2 n)$ použitím $O(n \log^2 n)$ komparátorov.*

Dôkaz 4.6.4: Hĺbka bitonickej triediacej siete na utriedenie ľubovoľnej postupnosti dĺžky $n = 2^k$ je $1 + 2 + ... + \log n = O(\log^2 n)$.

Počet komparátorov je daný vzťahom

$$\sum_{i=1}^{k-1} \frac{n}{2^i} C(2^i) = O(n \log^2 n).$$

□

Pozn.: Ďalšia dôležitá triediaca sieť je založená na "odd-even" spájacom algoritme. AKS triediaca sieť pracuje v hĺbke $O(\log n)$ a má veľkosť $O(n \log n)$.

Príklad 4.6.1: Nech $A = (a_1, a_2, ..., a_n)$, $B = (b_1, b_2, ..., b_n)$ sú utriedené postupnosti, pre $n = 2^k$. *"Odd-even" spájací algoritmus* pozostáva z rekurzívneho spájania dvoch "nepárnych" postupností $(a_1, a_3, ..., a_{n-1})$, $(b_1, b_3, ..., b_{n-1})$ do $(c_1, c_2, ..., c_n)$ a dvoch "párnych" postupností $(a_2, .a_4, ..., a_n)$, $(b_2, b_4, ..., b_n)$ do $(c'_1, c'_2, ..., c'_n)$. Potom je utriedená postupnosť daná ako $(c_1, \min\{c'_1, c_2\}, \max\{c'_1, c_2\}, \min\{c'_2, c_3\}, \max\{c'_2, c_3\}, ..., \min\{c'_{n-1}, c_n\}, \max\{c'_{n-1}, c_n\})$. Použitím $0-1$ princípu ukážeme správnosť "odd-even" spájacieho algoritmu. Odvodíme triediacu sieť založenú na "odd-even" spájacom algoritme a ???state jeho hĺbky a veľkosti. ♠

4.7 Problém výberu

Problém výberu: Je daná $A = (a_1, ..., a_n)$ a k, $1 \leq k \leq n$. Treba určiť $a_i \in A$ také, že $\texttt{rank}(a_i : A) = k$.

Špeciálne prípady:

- $k = 1$ (minimum), $k = n$ (maximum) — čas $O(\log \log n)$ na CRCW; $O(n)$ na EREW PRAM.
- $k = \lceil \frac{n}{2} \rceil$ (medián) — sekvenčný algoritmus pracujúci v lineárnom čase; čas $\Omega(\frac{\log n}{\log \log n})$ na CRCW PRAM

Všeobecný problém výberu: Používa $O(\log n)$-časový triediaci algoritmus s $O(n \log n)$ operáciami ... neoptimálne riešenie.

Optimálny paralelný algoritmus výberu možno získať použitím techniky "Accelerated Cascading" Redukujeme veľkosť n vstupného poľa na $O(\frac{n}{\log n})$.

Myšlienka redukcie veľkosti: Predpokladajme, že a z A je identifikované s $\frac{n}{4} \leq \texttt{rank}(a : A) \leq \frac{3n}{4}$. Potom a indukuje rozklad A na triedy A_1, A_2, A_3 prvkov $< a$, $= a$ a $> a$. Nech $s_i = |A_i|$. $s_1, s_3 \leq \frac{3n}{4}$. Ak k-ty najmenší prvok je v A_2, potom sme skončili. Inak zredukujeme veľkosť faktorom aspoň $\frac{3}{4}$. Opakovanie tohto procesu $O(\log\log n)$-krát zredukuje veľkosť A na $O(\frac{n}{\log n})$. Teraz môžeme použiť $O(\log n)$-časový triediac algoritmus používanjúci $O(n \log n)$ operácií na množinu s redukovanou veľkosťou.

Pozn.: pipelined merge-sort algoritmus ... čas $O(\log n)$, $O(n \log n)$ operácií.

Algoritmus 4.7.1: (Výber)

vstup: Pole $A = (a_1, ..., a_n)$ a celé kladné k $(1 \leq k \leq n)$ také, že $\log n$ je celé, ktoré delí n.

výstup: Prvok a_i z A taký, že $\texttt{rank}(a_i : A) = k$.

```
begin
 1: set n_0 := n; set s := 0;
 2: while n_s > n/log n do begin
      2.a: set s := s + 1;
      2.b: Rozdeľ A na bloky B_i, každý pozostávajúci z log n následných
           prvkov z A;
           Vypočítaj medián m_i každého bloku B_i;
      2.c: Vypočítaj medián a postupnosti (m_1, m_2, ..., m_{n/log n}) použitím
           pipelined spájacieho algoritmu;
      2.d: Urči počty s_1, s_2, s_3 prvkov menších, rovných, väčších ako a;
      2.e: if s_1 < k ≤ s_1 + s_2 then
             begin
               return(a); exit;
             end
           else if k ≤ s_1 then
             begin
               Stlač prvky A menšie ako a do následných miest;
               set n_s := s_1;
             end
           else if k > s_1 + s_2 then
             begin
               Stlač prvky A väčšie ako a do následných miest;
               set n_s := s_3; set k := k - (s_1 + s_3)[1]
             end;
    end
 3: Utrieď pole pozostávajúce z ostávajúcich n_s prvkov;
    return(k-ty prvok utriedeného poľa);
end.
```

[1] pozn.prekl.: v originály $k := k - (s_1 + s_{3_2})$ bol som z toho poriadny jeleň

Veta 4.7.1: *Algoritmus 4.7.1 správne vypočíta k-ty najmenší prvok vstupného poľa A. Tento algoritmus beží v čase $O(\log n \log\log n)$ použijúc lineárny počet operácií.*

Dôkaz 4.7.1:

krok 2.b: Implementovaný sekvenčným lineárne-časovým algoritmom pre výber. Každý blok vezme $O(\log n)$ času a $O(n_s)$ operácií.

krok 2.c: Používa pipelined merge-sort algoritmus v čase $O(\log n)$ a celkovo $O(\frac{n_s}{\log n}\log n) = O(n_s$ operácií.

krok 2.d: Označíme každý prvok a_i s 1, 2, 3 v závislosti na tom, či $a_i < a$, $a_i = a$ alebo $a_i > a$. Použijúc algoritmus 2.1.1 dostaneme s_1, s_2, s_3. Čas $O(\log n)$, $O(n_s)$ operácií.

krok 2.e: Čas $O(\log n)$, lineárny počet operácií.

Preto bude `while` slučka trvať $O(\log n)$ času s $O(n_s)$ operáciami. Kvôli nasledujúcej vete: $O(\log\log n)$ iterácií.

□

Veta 4.7.2: *Pre veľkosti zoznamov v následných iteráciách platí $n_s \leq \frac{3n_s}{4}$.*

Dôkaz 4.7.2: Samozrejme, stačí ukázať, že $\frac{n_s}{4} \leq \texttt{rank}(a : A_s) \leq \frac{3n_s}{4}$.

$A_0 = A$. Pretože a je medián z prvkov m_i, a je väčšie ako polovica z prvkov m_i. Ale m_i je väčšie ako $\frac{1}{2}\log n$ prvkov v B_i, pre $\forall i$. Preto je a väčšie alebo rovné $\frac{n_s}{2\log n}\frac{\log n}{2} = \frac{n_s}{4}$ prvkom z A_s.

□

krok 2: čas $O(\log n \log\log n)$, $O(\sum n_s) = O(n)$ operácií.

krok 3: čas $O(\log n)$, $O(n)$ operácií. EREW PRAM

4.8 Resumé

Algoritmus	Čas	PRAM model
4.1.1 Paralelné vyhľadávanie	$O\left(\frac{\log(n+1)}{\log(p+1)}\right)$	CREW
4.3.1 Ohodnocovanie utriedenej postupnosti v ďalšej utriedenej postupnosti	$O\left(\frac{(n+m)\log\log m}{p} + \log\log m\right)$	CREW
Optimálne spájanie	$O\left(\frac{n}{p} + \log\log n\right)$	CREW
4.5.1 Jednoduchý merge-sort	$O\left(\frac{n\log n}{p} + \log n \log\log n\right)$	CREW
Pipelined merge-sort	$O\left(\frac{n\log n}{p} + \log n\right)$	CREW
4.6.1 Bitonické spájanie	$O\left(\frac{n\log n}{p} + \log n\right)$	EREW
4.7.1 Výber	$O\left(\frac{n}{p} + \log n \log\log n\right)$	EREW

Dolné hranice na modeli stromu paralelného porovnávania

Problém	Dolná hranica	Počet procesorov
Vyhľadávanie	$\Omega\left(\frac{\log n}{\log(p+1)}\right)$	$p \leq n$
Maximum	$\Omega\left(\frac{n}{p} + \log\log n\right)$	$p \leq n$
Spájanie	$\Omega\left(\frac{n}{p} + \log\log n\right)$	$p \leq n\log^k n$ pre konštantné k

5. Efektívne paralelné algoritmy

5.1 Model paralelného výpočtu

tuto ma prist obrazok ...

5.2 Redukcia počtu procesorov

Veta 5.2.1: *Nech A je daný algoritmus s paralelným časom výpočtu t. Predpokladajme, že A vyžaduje celkový počet m výpočtových operácií. Potom A možno implmentovať použijúc p procesorov v paralelnom čase $O(\frac{m}{p} + t)$.*

Dôkaz 5.2.1: Nech $m(i)$ je počet operácií vykonaných (paralelne) v kroku i algoritmu A. Použitím p procesorov toto možno odsimulovať v čase $\left\lfloor \frac{m(i)}{4} \right\rfloor + 1$.

$$\sum_{1 \leq i \leq t} \left(\frac{m(i)}{p} + 1 \right) = \frac{m}{p} + t$$

$(m = m(1) + ... + m(t))$ □

6. Paralelný string-matching

Nech `pat` a `text` sú stringy dĺžky m, n. `pat` sa vyskytuje na pozícii i v stringu `text` $\Longleftrightarrow$ `pat` = `text`$[i..i+m+1]$. Jednoduchý algoritmus pracuje v čase $O(\log m)$ s $n\frac{m}{\log n}$ procesormi na PRAM. Procesory organizované v binárnom strome na výpočet `boolean and` z m hodnôt `match`(i) = `text`$[i+k-1]$ = `pat`$[k]$ pre $k = 1, ..., m$.

Algoritmus nie je optimálny, pretože $pt = O(n^2)$.

- Galil, 1984: $O(\log^2 n)$ času, $\frac{n}{\log^2 n}$ procesorov, PRAM
- Vishkin, 1985: veľkosť abecedy nie je pevná

Majme dané stringy u, v. u je *periódou* v, ak v je prefixom u^k. *Veľkosť periódy* v je dĺžka najkratšieho bloku v. (*svedok*): Pre dané j predpokladajme `pat`$[j..m]$ nie je prefixom `pat` (`pat`$[j..m] \neq$ `pat`$[1..m-j+1]$). Potom existuje celé w $(1 \leq w \leq m-j+1)$ také, že `pat`$[w] \neq$ `pat`$[s]$, $s = j-1+w$. w je *svedkom* tejto nezhody.

$$\begin{array}{cccccccccc} & & & & 1 & — & w & — & – & – & – \\ 1 & — & j & — & s & — & – & – & – & & \end{array}$$

Nech `wit`$[j]$ je ľubovoľný svedok w. `pat` = *abaababa*, `wit`$[1] = 0$, `wit`$[2] = 1$, `wit`$[3] = 2$, `wit`$[4] = 4$.
Vzorka je *aperiodická* $\Longleftrightarrow$ `wit`$[j] \neq 0$ pre každé $2 \leq j \leq \frac{m}{2}$
$\Longleftrightarrow$ veľkosť jeho najmenšej periódy je $\geq \frac{m}{2}$.
Vzorka `pat` = *abaababa* je aperiodická.

6.1 Analýza textu

Aperiodický prípad

Predpokladajme, že `wit` je známe pre $1 \leq j \leq \frac{m}{2}$. Ako možno rýchlo nájsť `pat` v `text` ?

Nech $n' = n - m + 1$. `wit1`$[i]$ je svedkom nezhody, ak sa vzorka umiestnená na pozíciu i nezhoduje s textom; inak `wit1`$[i] = 0$ (kandidáti pre zhodu).

Duel medzi dvomi pozíciami p, q: $1 \leq p \leq q \leq n'$, $q - p < \frac{m}{2}$. Výskyt `pat` nemôže začínať aj na p aj na q.

Nech $j = q - p + 1$, `wit`$[j] = w$. Predpokladajme, že `text`$[p..p+m-1]$ = `pat`. Potom {def `wit`} `pat`$[j-1+w] \neq$ `pat`$[w] \Longrightarrow$ `text`$[q+w-1] \neq$ `pat`$[w] \Longrightarrow$ `pat` sa nevyskutuje na $q \Longrightarrow$ `wit1`$[p] = q - p + w$.

`pat`$[w]$ = `text`$[q+w-1]$ a `text`$[p+q-p+w-1] \neq$ `pat`$[q-p+w] \Longrightarrow$ `pat` sa nevyskytuje na $p \Longrightarrow$ `wit1`$[p] = q - p + w$.

$$\begin{array}{ccccccccccccc} & & & & 1 & — & w & — & – & – & – & \texttt{pat (nad } q\texttt{)} \\ & & 1 & — & j & — & (j-1+w) & — & – & – & – & \texttt{pat (nad } p\texttt{)} \\ 1 & — & p & — & q & — & q+w-1 & — & – & – & – & \texttt{text} \end{array}$$

```
function Duel(p, q)                                          {wit[q − p + 1] ≠ 0}
begin
    if p = null then Duel := q else
        if q = null then Duel := p else begin
            set j := q − p + 1;
            set w := wit[j];
            if text[q + w − 1] ≠ pat[w]
                then begin
                  set wit1[q] := w;
                  set Duel := p;
                end else begin
                  set wit1[p] := q − p + w;
                  set Duel := q;
                end;
        end;
end;
```

Predpokladajme, že m, n' sú mocniny 2. k-ty rozklad $[1..2^k][2^k+1..2\times 2^k]...[i2^k+1..(i+1)2^k]...$ pre $k = 0, 1, ..., \log m - 1$. V každom k-bloku existuje najviac jedna pozícia i, kde `pat` začína. Na začiatku nech $k = 0$, $\texttt{wit1}[i] = 0$ pre každé i. Ak $k = k+1$, potom k-blok má dvoch kandidátov $\Longrightarrow$ duel. Iterujme túto procedúru $(\log m - 1)$-krát. `wit1` je *k-riedky*[1] práve vtedy, keď každý k-blok má najviac jednu položku j s $\texttt{wit1}[j] \overset{\neq 0}{=} 0$.[2]

```
procedure MoreSparse(text, k)                                //wit1 je k-riedke;
begin
    for ∀k-bloky [3]pardo begin
        Nech p, q sú dvaja kandidáti v tomto bloku;
        // položky s nulovými hodnotami wit1
        candidate := Duel(p, q);
        // nech r je eliminované v tomto bloku; teraz wit1[r] ≠ 0
    end;
end;

procedure MakeSparse(text, sparsity);
begin
    for k := 1 to sparsity do
        MoreSpace(text, k);
end;
```

MakeSparse(`text`, $\log m - 1$) $\quad O(n) = n + \frac{n}{2} + \frac{n}{4} + ...$ operácií
$\log m$ hĺbka

[1] IN ENGLISH: k-sparse
[2] pozn.prekl.: ???
[3] poz.prekl.: v origináli "in parallel `do` " ... takže teraz neviem

Brentova veta: čas $O(\log m)$, $O(\frac{n}{\log m})$ procesorov ... PRAM.

Lema 6.1.1: *V tomto aperiodickom prípade a s predpokladom, že* `wit` *je predpočítané pre danú vzorku, string-matching možno vykonať v čase* $O(\log m)$ *s* $\frac{n}{\log m}$ *procesormi na* PRAM.

Nech `period`(v) je najkratšia perióda v. `periodsize`(v) označíme $|$`period`$(v)|$. Reťazec v je periodický práve vtedy, keď $|v| \geq 2$`periodsize`(v).

Lema 6.1.2: (lema o periodicite)

Ak v *má dve periódy veľkostí* p, q *a* $|v| \geq p + q$, *potom* v *má tiež periódu veľkosti* $\gcd(p, q)$.

Dôkaz 6.1.2: Nech $p > q$. Ak v má periódy veľkostí p, q ($|v| \geq p + q$), potom v má tiež periódu veľkosti $p - q$ ... $\forall i : v[i] = v[i+p] \wedge v[i] = v[i+q] \Longrightarrow v[i+p] = v[i+q] \Longrightarrow \forall i'; i' = i - q : v[i'] = v[i' + (p-q)]$, $|v| \geq p + q$ □

Euklidovská metóda pre výpočet $\gcd(p, q)$

```
function gcd(p,q);
begin
    set p' := p; set q' := q;
    while p' ≠ q' do                                    //invariant
        if p' > q' then set p' := p' - q' else set q' := q' - p';
    set gcd := p';
end;
```

Invariant: v má periódy veľkostí p', q'. Ak sú posledné hodnoty p', q' rovné $\gcd(p, q)$, potom v má periódu veľkosti $\gcd(p, q)$.

Predpokladajme, že `pat` je periodická s `periodsize`$($`pat`$) - 7$.[4]

Lema 6.1.3:

(1) *Ak sa* `pat` *nachádza na dvoch rôznych pozíciách* i, j, *potom* $|i - j| \geq P$.

(2) *Ak sa* `pat` *nachádza na* j *a* $j + D$, *kde* $D \leq m - P$, *potom* D *je násobkom* P.

(3) *Ak sa* `pat` *nachádza na* j *a* $j + D$, *kde* $0 < D \leq \frac{m}{2}$, *potom sa* `pat` *nachádza aj na* $j + P$.

Nech `period`$($`pat`$) = w,$`pat` $= u^s x, |w| = P, |x| \leq P$. Vezmime `pat`$' = ux$. Vzorka `pat`$'$ je aperiodická. Všetky výskyty `pat`$'$ v `text` nájdeme v čase $O(\log m)$ použitím $\frac{n}{\log m}$ procesorov na PRAM s predpokladom, že `wit` je predpočítané.

- `next`$(i)=i + P$: ak sa vzorka `pat`$'$ nachádza na i
  `next`$(i)=i$: inak
- pre `next`$(i)\neq i$: `largest`$(i) = \max\{k \mid$ `next`$^k(i) \neq$ `next`$^{k-1}(i)\}$
  `next`$(i)=i$: `largest`$(i) = 0$

Ak sa vypočíta `largest`, potom možno určiť všetky výskyty `pat`. {`pat` sa nachádza na pozícii $i \Longleftrightarrow$ `largest`$(i) \geq s\}$ `largest` vypočítané v čase $O(\log n)$ s $\frac{n}{\log m}$ procesormi na PRAM, ak je dané `match`$($`pat`$')$.[5]

[4] pozn.prekl.: ???
[5] pozn.prekl.: ???

Ako vypočítať `largest` ?

Dekomponujme `match`(`pat`$'$) do P podpostupností. {p-ta postupnosť: pozície i s $((i-1) \bmod P = p)$ pre $p = 0, ..., P-1$}.

Pre každú podpostupnosť pre každú pozíciu obsahujúcu 1 vypočítame počet následných 1tiek z tejto pozície napravo.

Dĺžka podpostupnosti ... $\frac{n}{p} \Longrightarrow O(\log \frac{n}{p})$ času s $\frac{\frac{n}{p}}{\log \frac{n}{p}}$ procesormi. P podpostupností ... $O(\log \frac{n}{p})$ času s $\frac{n}{\log \frac{n}{p}}$ procesormi. Nakoniec, $\log n \geq \log \frac{n}{p}$, takže môžeme zoskupiť operácie do skupiniek veľkosti $\frac{\log n}{\log \frac{n}{p}}$.

Lema 6.1.4: *Ak bola vzorka predspracovaná a* `wit` *vypočítané, potom pre periodický prípad možno string-matching vykonať v čase* $O(\log n)$ *s* $\frac{n}{\log n}$ *procesormi.*

Kombinované výsledky pre analýzu textu

Lemy 6.1.1+6.1.4: v 6.1.1 výmenný čas pre procesory {$\log n \geq \log m$ redukcia procesora faktorom $\frac{\log m}{\log n}$}[6]

Veta 6.1.5: *Ak je vzorka predspracovaná a* `wit` *vypočítané, potom string-matching možno vykonať v čase* $O(\log n)$ *s* $\frac{n}{\log n}$ *procesormi na* PRAM.

6.2 Predspracovanie vzorky

Vypočítame `wit` pre segment $[1..2^{\log m - 1}]$; `wit`$(1) = 0$

$$\texttt{witness}(j,r) = \begin{cases} 0 & \text{ak } \texttt{pat}[j..r] \text{ je prefixom } \texttt{pat} \\ k & 1 \leq k \leq r-j+1 \text{ také, že } \texttt{pat}[k] \neq \texttt{pat}[j+r-1] \end{cases} \quad \texttt{pat}[k] \neq \texttt{pat}[j+k-1]$$

	1	—	k	—	–	–	–	vzorka
—	j	—	$j+k-1$	—	r			text (=vzorka)

a teda `witness`$(j,r) = k$. `witness`(j,r) je vypočítané v čase $O(\log r)$ s $O(\frac{r}{\log n})$ procesormi na PRAM.

No ...

... ďalej sa mi to už prekladať nechce. Ak sa to niekomu chce doprekladať, nech mi mailne na adresu uvedenú na titulnej strane. Po dohode mu môžem poskytnúť zdrojáky.

Zbohom !

[6] pozn.prekl.: ???