Přednáška #7: **Paralelní prefixový součet a technika eulerovské cesty**

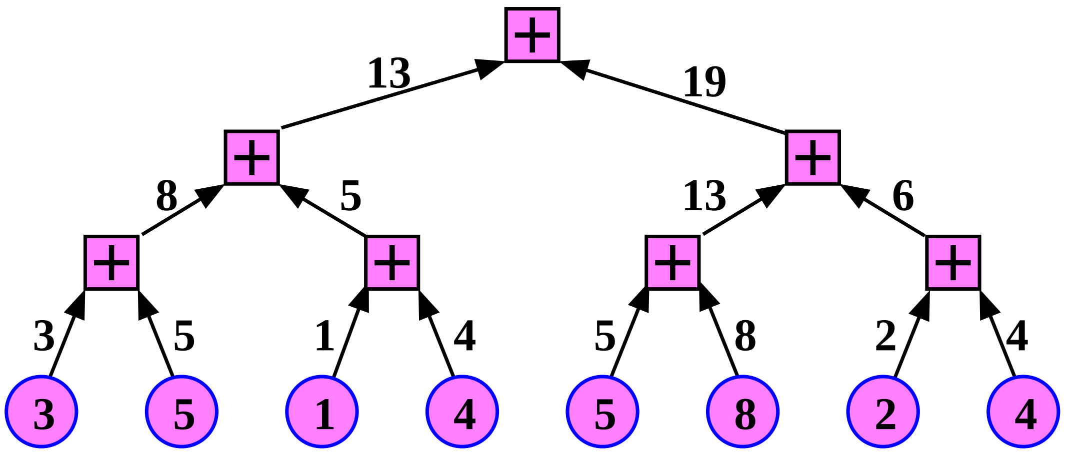
Paralelní redukce

- Vstupní pole X n hodnot $x_0, x_1, .., x_{n-1}$ z množiny D.
- Vypočti
$$S = x_0 \oplus x_1 \oplus \cdots \oplus x_{n-1},$$
kde $\oplus$ = binární operace nad množinou D.
- Redukce je krásně paralelizovatelná $\iff$ $\oplus$ = asociativní.

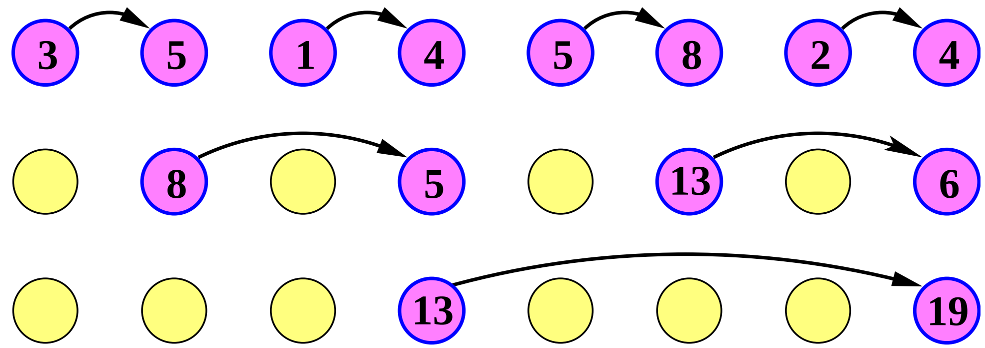
Vlastnosti paralelní redukce

- $T(n,p) = \alpha n/p + \beta \log p$.
- $C(n,p) = \alpha n + \beta p \log p$.
- $E(n,p) = \alpha n / \alpha n + \beta p \log p))$.
- Dobrá škálovatelnost: $\psi_1(p) = p \log p$ a $\psi_2(n) = n/\log n$.
- Časová optimalita: $T_{\min}(n,p) = O(\log n)$ a $\psi_3(n) = n/\log n$.
- Je to normální hyperkubický algoritmus $\Longrightarrow$ optimální na hyperkubických sítích.

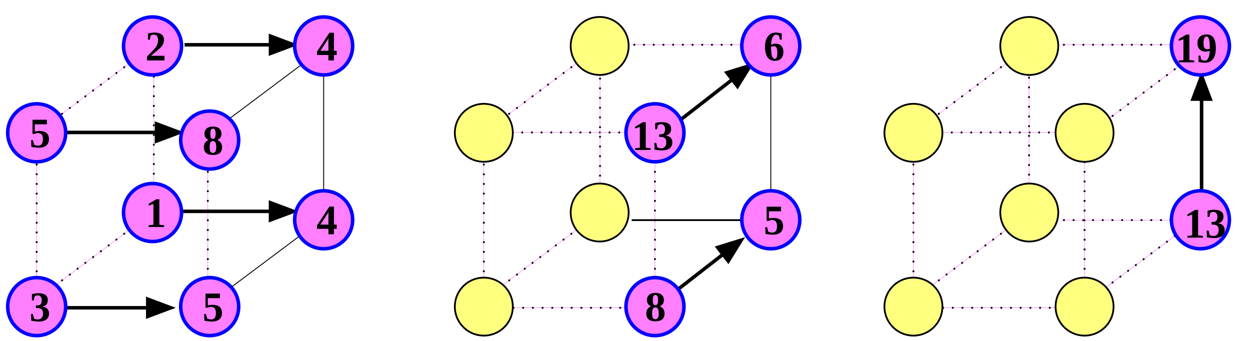
Implementace paralelní redukce



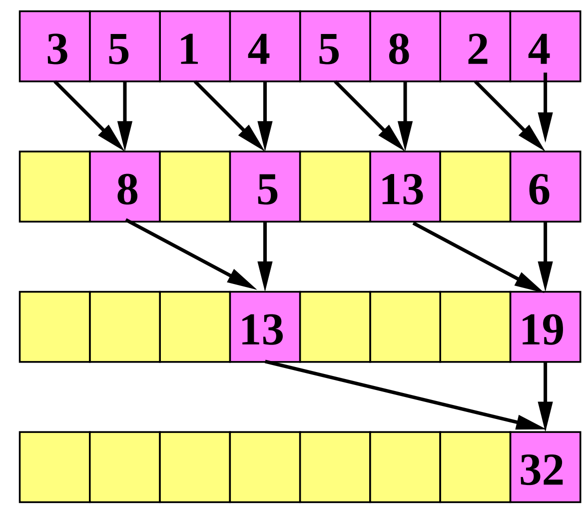
(a) (Neprimy) uplny binarni strom



(c) WH 1−D mrizka



(b) Uplny graf a SF hyperkrychle



(d) EREW PRAM

Paralelní prefixový součet na poli (PPS)

- Vstupní pole $X = x_0, .., x_{n-1}$ z množiny D.
- Asociativní binární operátor $\oplus$ v D.
- Úkolem je vypočítat pole Y všech prefixů pole X: $y_i = x_0 \oplus x_1 \oplus \ldots \oplus x_i$.

2	**4**	**1**	**5**	**3**	**9**	**8**	**3**	**2**

2	**6**	**7**	**12**	**15**	**24**	**32**	**35**	**37**

- Sekvenční algoritmus:

Algorithm $\textsc{PrefixSum}$(**in:** $X[0, \ldots, n-1]$; **out:** $Y[0, \ldots, n-1]$;)

$\{$ **var** i :`int`,$sum : \mathcal{D}$;

$i := 0; sum := X[i]; Y[i] := sum;$

while $(i < n - 1)$ **do**

$\{\ i := i + 1;\ sum := sum \oplus X[i];\ Y[i] := sum\}$

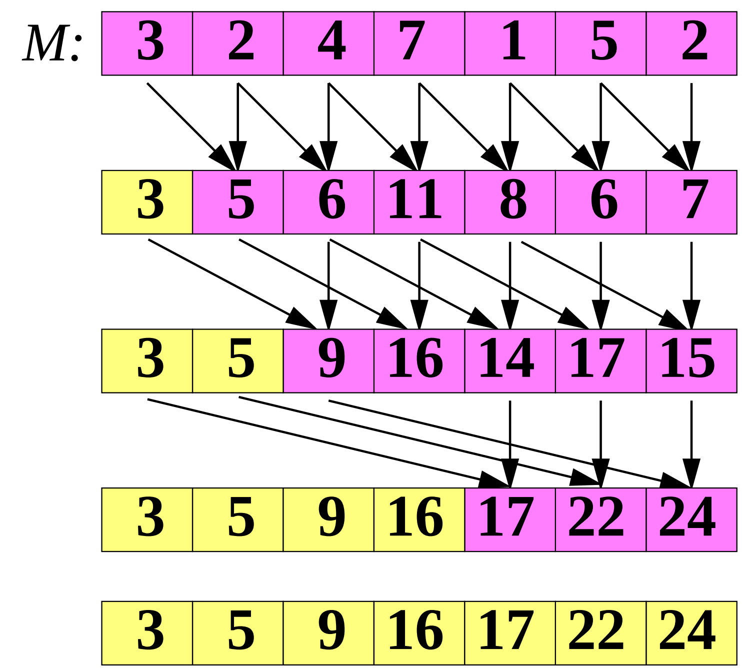
$\}$

PPS na EREW PRAM

- Na počátku: $P_0, .., P_{n-1}$ a $M[0], .., M[n-1]$ obsahující po řadě $x_0, .., x_{n-1}$.
- P_i má registr y_i.
- Krok j: $M[k] = x_k \oplus x_{k-1} \oplus \cdots \oplus x_{k-2^{j+1}+1}$.
- Táž časová složitost jako EREW PRAM paralelní redukce!!!!!

Alg. $\mathrm{PRAM\_PPS}$(**in,out:** $M[0,\ldots,n-1]$);

for all $i := 0,\ldots,n-1$ **do_in_parallel**
  $y_i := M[i]$;
for $j := 0,\ldots,\lceil \log n \rceil - 1$ **do_sequentially**
  { **for all** $i := 2^j,\ldots,n-1$ **do_in_parallel**
    $y_i := y_i + M[i-2^j]$;
    for all $i := 2^j,\ldots,n-1$ **do_in_parallel**
    $M[i] := y_i$ }

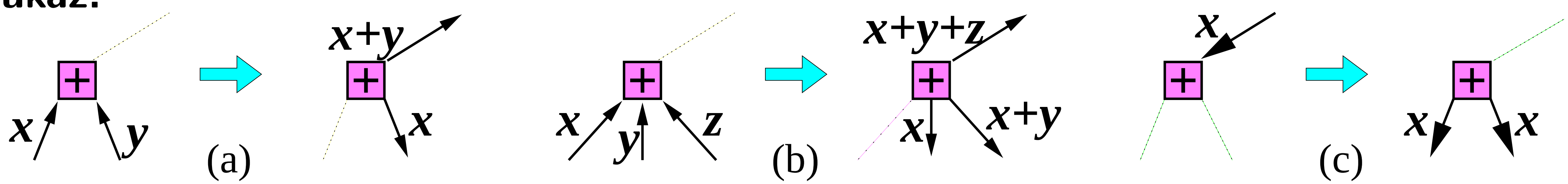


Generický PPS na nepřímém stromu

Definice 1. *Nepřímý strom: Vstupní data pouze v listech, vnitřní uzly pouze počítají.*

Lemma 2. *PPS n vstupních hodnot lze řešit na binárním nepřímém stromu T s n listy v $2h(T)$ krocích, kde $h(T)$ je výška T.*

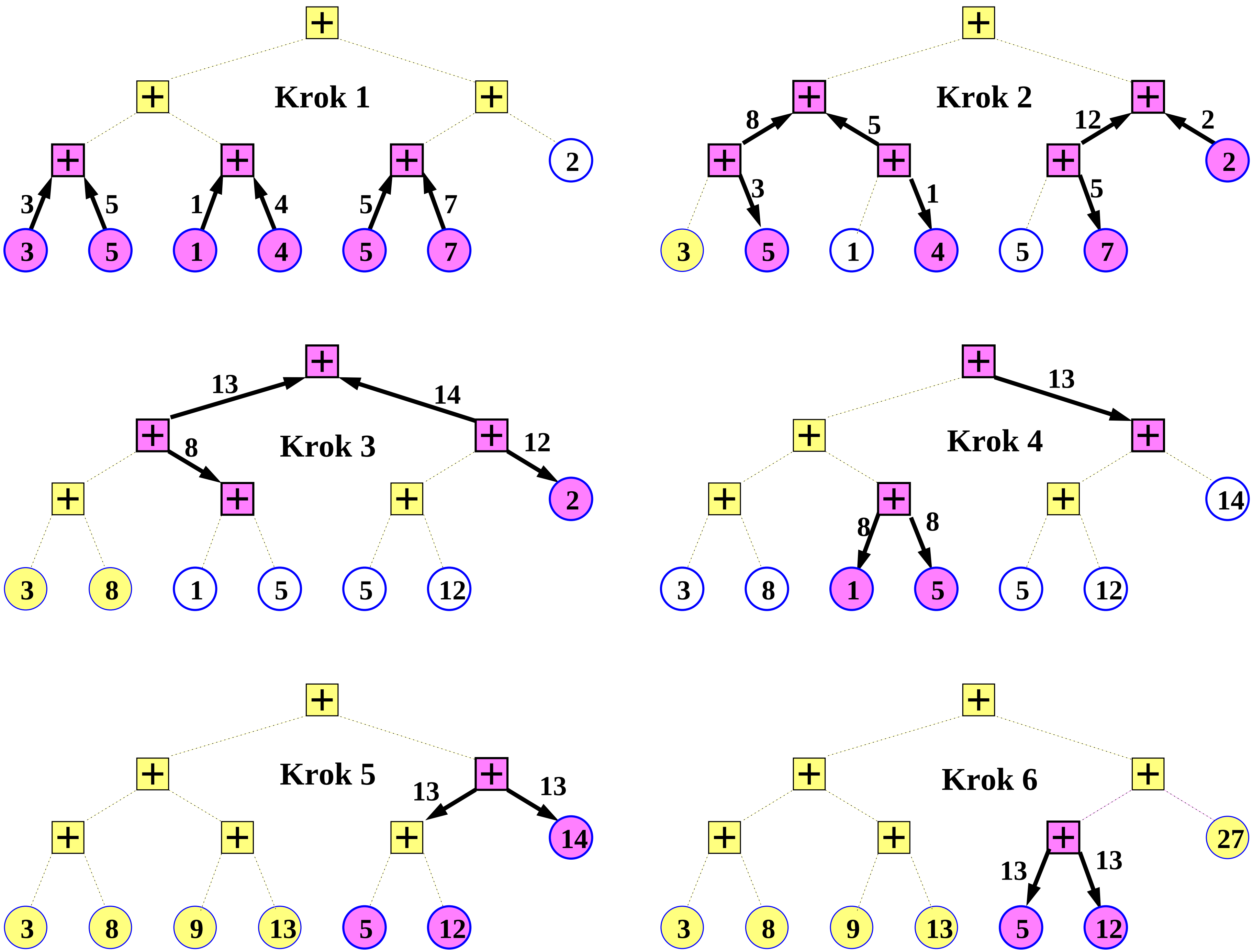
Důkaz.



1 vzestupná vlna (a), (b) spouští $h(T)$ sestupných vln (c). ♣

Důsledek 3. *T = úplný binární strom $\implies$ $O(\log n)$ kroků.*

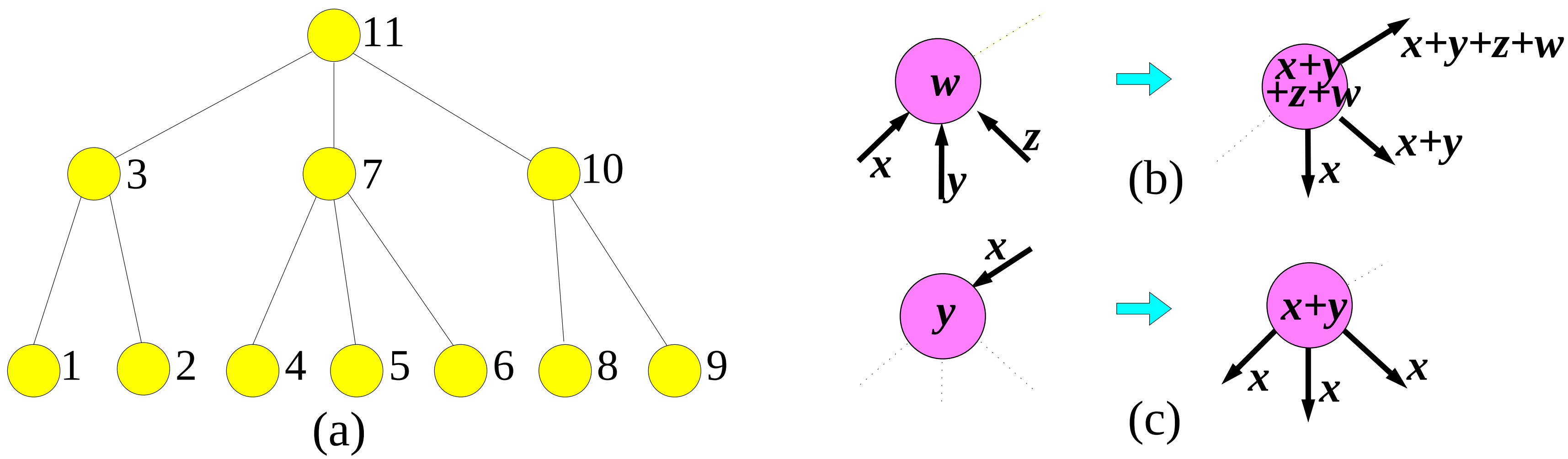
Příklad PPS na nepřímém stromu



Generický PPS na přímém stromu

Lemma 4. *Uvažujme <u>přímý</u> n-uzlový strom T omezeného stupně větvení a výšky $h(T)$, na který je v pořadí POSTORDER namapováno pole A n vstupních hodnot. Pak PPS na poli A lze vypočítat v $2h(T)$ krocích.*

Důkaz.



PPS na libovolné topologii

Lemma 5. *PPS n vstupních hodnot lze řešit na <u>libovolné</u> n-uzlové síti G s omezeným stupněm a s průměrem $\mathrm{diam}(G) = \Omega(\log n)$ v čase $O(\mathrm{diam}(G))$.*

Důkaz.

1. konstrukce kostry do šířky (hloubka = excentricita kořenu $\leq \mathrm{diam}(G)$),
2. aplikace předchozího algoritmu. ♣

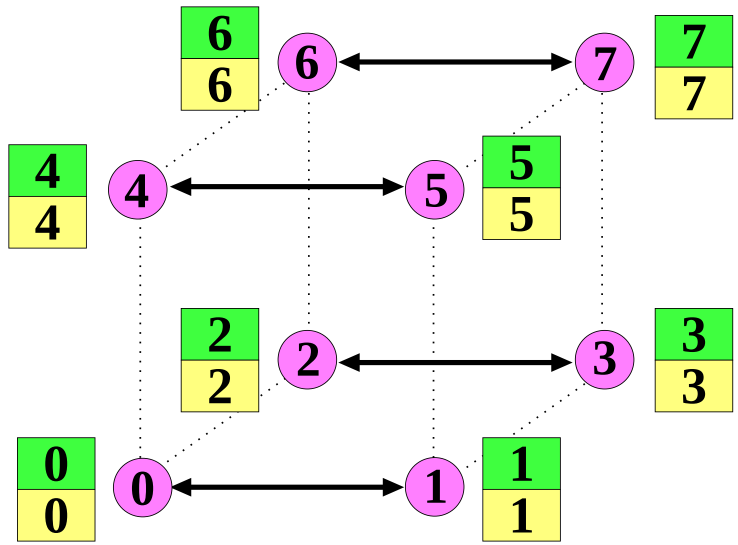
PPS na hyperkrychli

- Uvažujme Q_r, $r \geq 1$.
- Paralelní prefixový součet = normální hyperkubický algoritmus.
- PPS = jednoduché rozšíření algoritmu pro All-to-All Broadcast.
- Každý procesor P_i, $i = 0, \ldots, 2^r - 1$, má 2 pomocné registry, $zeleny_i$ a $zluty_i$.

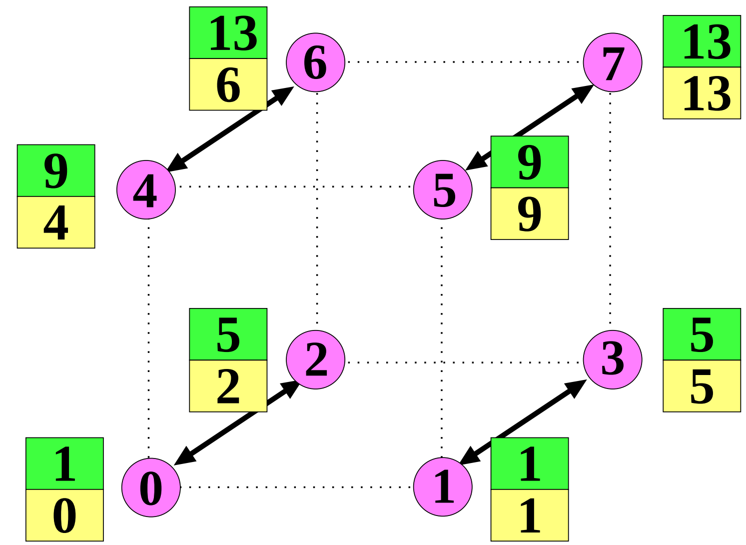
Algorithm $\textsc{Hypercube\_PPS}(X[1, \ldots, n-1])$

all $P_i, i := 0, \ldots, 2^r - 1$, **do_in_parallel**
  $\{$ $zeleny_i := zluty_i := X[i]$;
   for $j := 0, \ldots, r-1$ **do_sequentially**
    $\{$ **send** $zeleny_i$ **to** $P_{i \,\mathrm{XOR}\, 2^j}$;
     receive $novyzeleny$ **from** $P_{i \,\mathrm{XOR}\, 2^j}$;
     $zeleny_i := zeleny_i + novyzeleny$;
     if $(i \,\mathrm{XOR}\, 2^j < i)$ **then** $zluty_i := zluty_i + novyzeleny$ $\}\}$

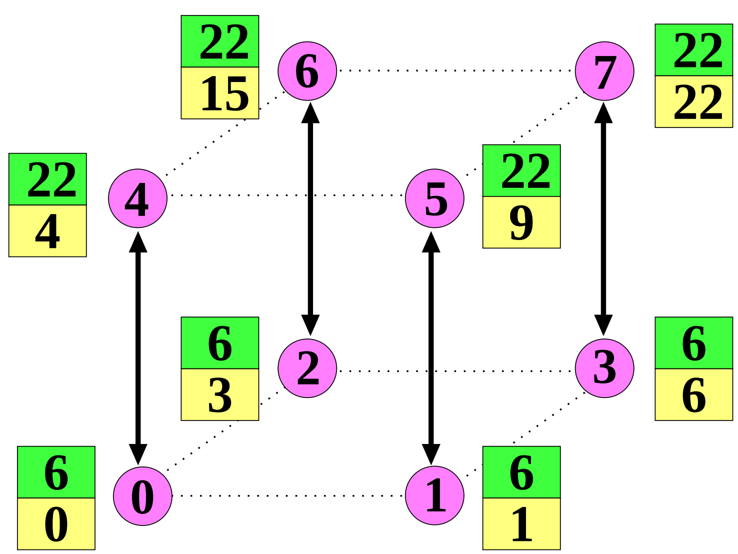
Příklad hyperkubického PPS



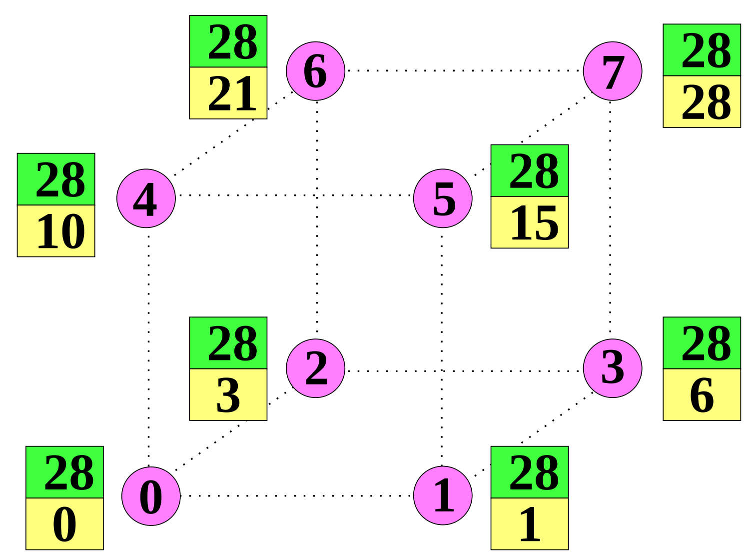
(a) Pocatecni stav



(b) Po provedeni kroku 1

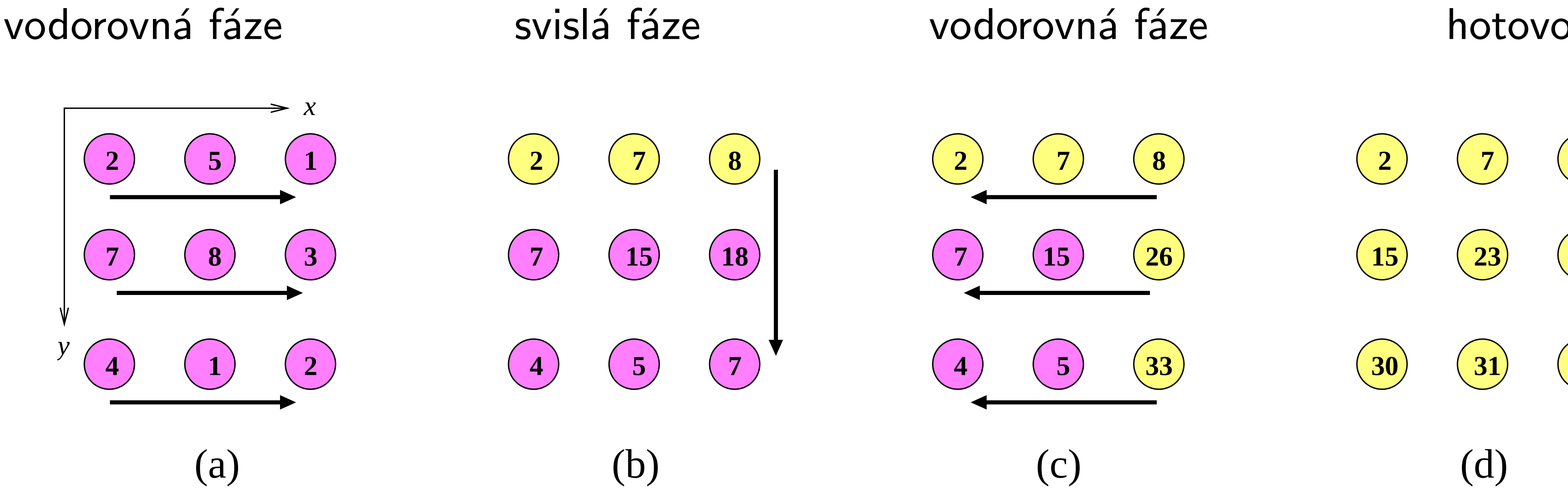


(c) Po provedeni kroku 2

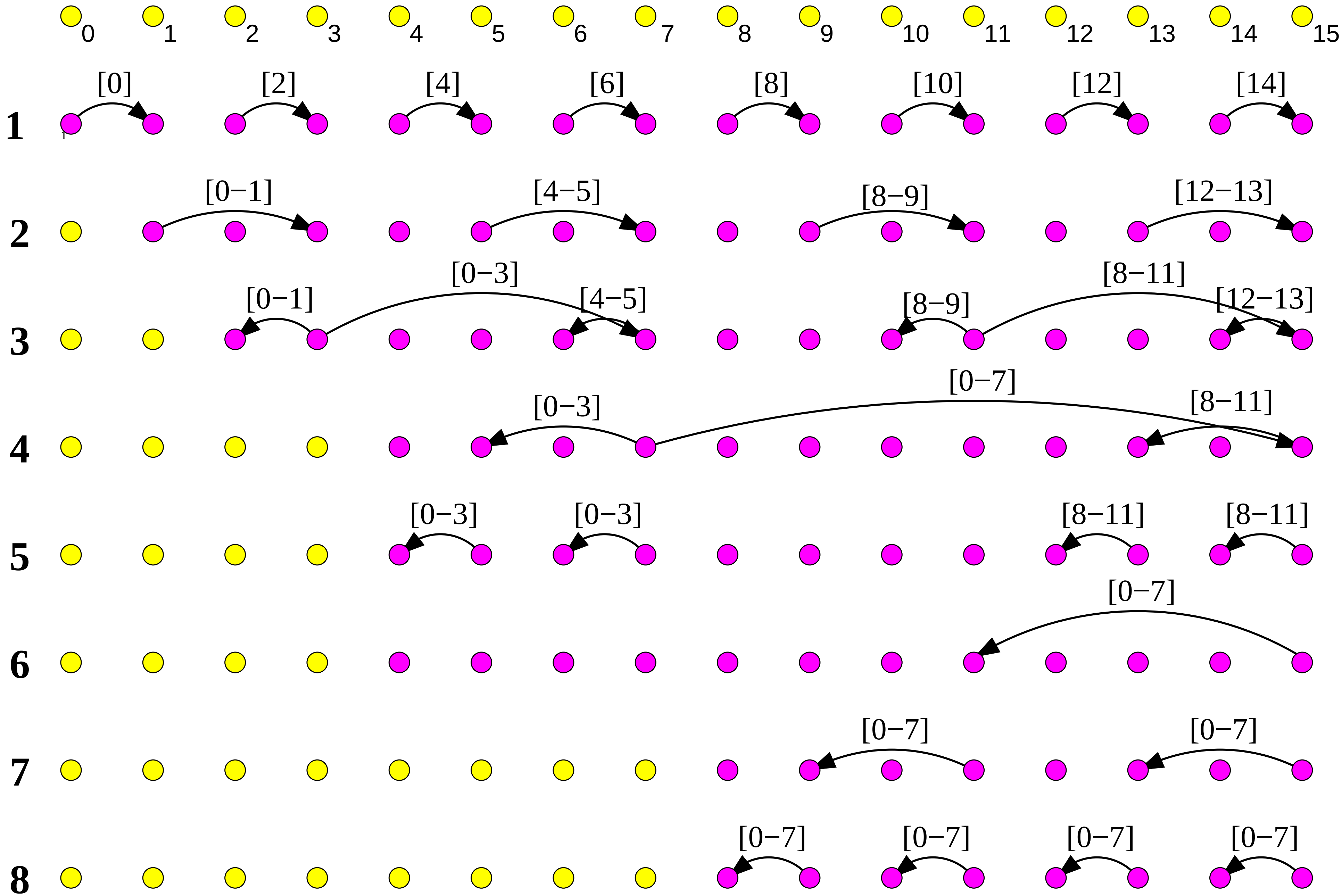


(d) Po provedeni kroku 3

PPS na SF mřížkách



Základní PPS algoritmus na 1-D 2-portové mřížce



Škálovatelnost paralelního prefixového součtu

1. Vstupní pole $X = \{x_0, \ldots, x_{n-1}\}$ a p procesorů $P_0, \ldots, P_{p-1}$.
2. Nechť $q = n/p$ a X je rozděleno do subpolí $X_0, \ldots, X_{p-1}$.

Algorithm $\textsc{Scaled\_PPS}(X[1, \ldots, n-1])$

---

for all $i := 0, \ldots, p-1$ **do_in_parallel**
 { P_i sekv. vypočte pole $S_i = [s_{i,0}, \ldots, s_{i,q-1}]$ = prefix součet X_i;
 (*$O(q)$ kroků.*)
 Definuj $z_i := s_{i,q-1}$. }
všechny Ps provedou PPS na $Z = [z_0, \ldots, z_{p-1}]$ a vygenerují $[\sigma_0, \ldots, \sigma_{p-1}]$;
 (*$O(\log p)$ kroků. *)
(*σ_i = celkový součet všech čísel na procesorech $P_0, \ldots, P_i$.*)
for all $i := 0, \ldots, p-2$ **do_in_parallel** P_i pošle σ_i P_{i+1};
for all $i := 1, \ldots, p-1$ **do_in_parallel** P_i přijme σ_{i-1} od P_{i-1};
for all $i := 1, \ldots, p-1$ **do_in_parallel** P_i připočte σ_{i-1} ke všem prvkům S_i;
 (* $O(q)$ kroků. *)

Příklad s 5 procesory

	P_0			P_1			P_2			P_3			P_4		
pocatecni rozdeleni	2	1	3	1	2	0	4	2	3	5	0	3	1	4	2
lokalni prefixove soucty	2	3	6	1	3	3	4	6	9	5	5	8	1	5	7
paralelni globalni prefix.soucet			6			9			18			26			33
posun doleva				+			+			+			+		
dopocitani lokalnich souctu	2	3	6	7	9	9	13	15	18	23	23	26	27	31	33

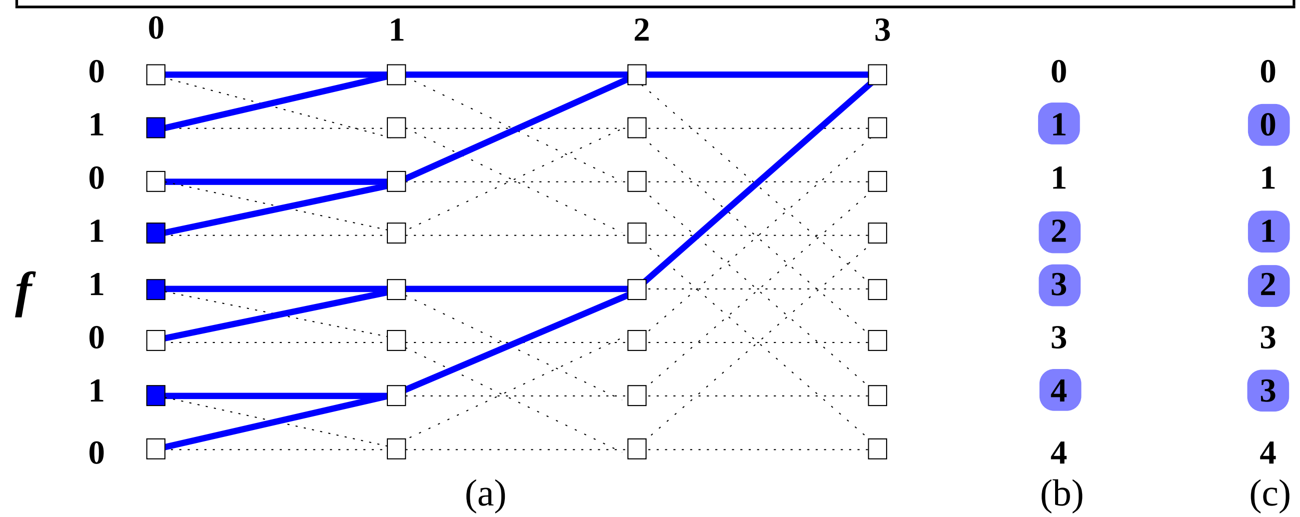
- $T(n,p) = O(n/p + \log p)$ kroků na PRAMech, hyperkrychli, WH mřížkách a sítích s $O(\log p)$ průměrem.
- Paralelní prefixový součet má tutéž škálovatelnost jako paralelní redukce!!!!!
- Opět je to normální hyperkubický algoritmus $\Longrightarrow$ optimální na hyperkubických sítích.

Aplikace paralelního prefixového součtu

Zhušťovací problém

1. Každý vstupní uzel má příznak f, nastavený buď na 0 nebo 1.
2. Nejprve je nutno vypočítat index v rámci distribuované podmnožiny $\mathcal{M}$ uzlů s $f = 1$.
3. Pak je paket z i-tého uzlu v rámci $\mathcal{M}$ poslán na výstup číslo i.

Zhušťovací problém na nepřímém motýlku $indBF_3$



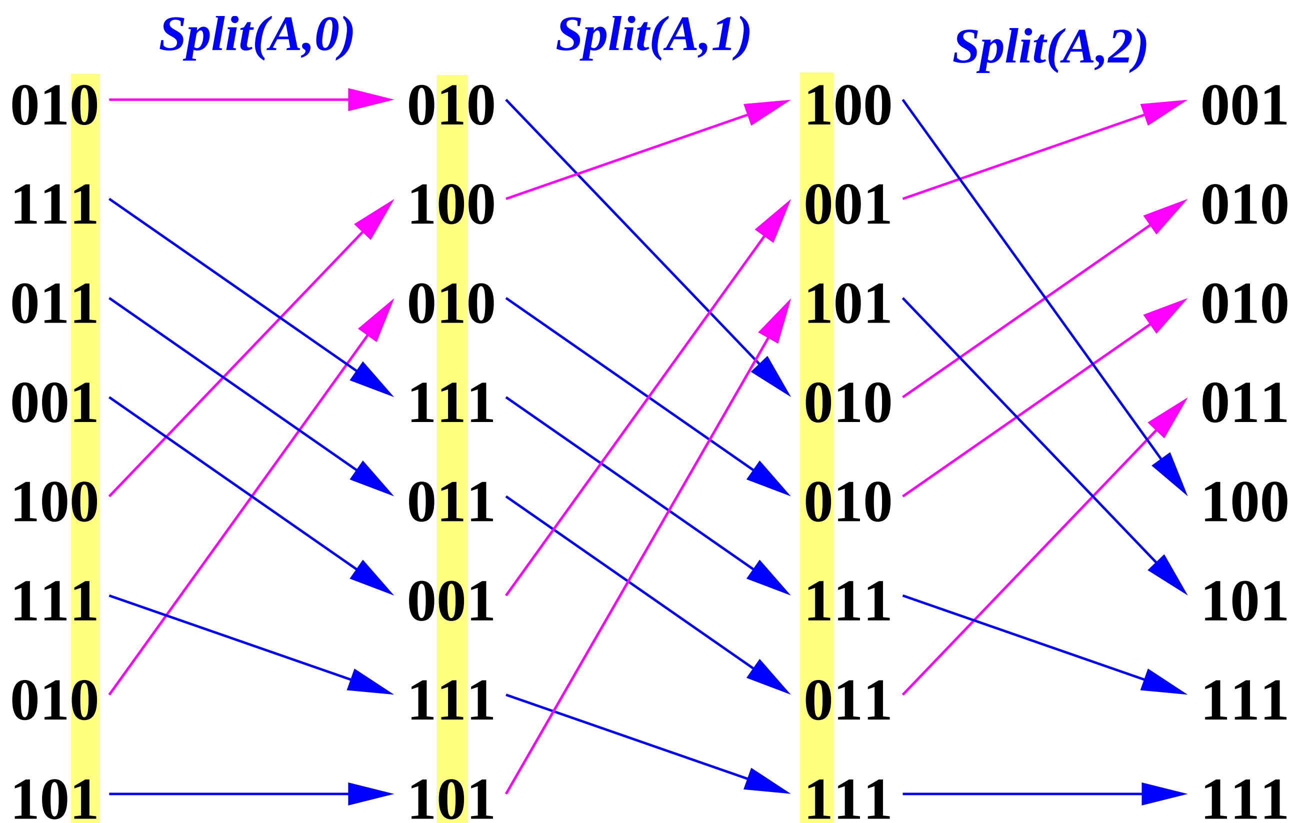
(a) Počáteční hodnoty příznaků a jeden možný nepřímý strom.
(b) Pole příznaků po paralelním prefixovém součtu.
(c) Konečné hodnoty indexů cílových řádků.

Algorithm $\textsc{RadixSort}(A[0,\ldots,2^n-1])$
for $i := 0,\ldots,n-1$ **do_sequentially** $Split(A,i)$

- $Split(A,i)$ = permutace A: všechna čísla, jejichž bit i je 0, jsou vytlačena nahoru a všechna čísla, jejichž bit i je 1, jsou stlačena dolů.

  $\Longrightarrow$ $Split(A,i)$ = 2 paralelní prefixové součty + 2 zhušťovací permutační směrování.

- Paralelní časová složitost RadixSortu 2^n čísel na oBF_n je $O(n^2)$.



Paralelní binární sčítačka s predikcí přenosu

- Vstup: 2 n-bitová binární čísla $X = x_{n-1} \dots x_0$ a $Y = y_{n-1} \dots y_0$.
- Jak vypočítat $Z = z_{n-1} \dots z_0 = X + Y$ v $\log n$ krocích? Předpočítáním přenosových bitů.
- Výstup: $C = c_n, \dots, c_1, c_0$, kde vždy $c_0 = 0$.

for $i := 0, \dots, n-1$ **do_in_parallel**

$$b_i := \begin{cases} g(enerate) & \textbf{if } x_i = y_i = 1, \\ s(top) & \textbf{if } x_i = y_i = 0, \\ p(ropagate) & \textbf{jinak}; \end{cases}$$

vypočti $B' = [b'_{n-1}, \dots, b'_0]$ použitím
PPS $\odot$ nad polem $[b_{n-1}, \dots, b_0, s]$

kde

$\odot$	s	p	g
s	s	s	s
p	s	p	g
g	g	g	g

;

for $i := 1, \dots, n$ **do_in_parallel**
$c_i := 1 \iff b'_{i-1} = g$;

X:		0	1	0	1	1	0	1	1	
Y:		0	1	1	0	1	0	0	1	
B:		s	g	p	p	g	s	p	g	s
B':		s	g	g	g	g	s	g	g	
C:	0	1	1	1	1	0	1	1	0	
Z:		1	1	0	0	0	1	0	0	

Tridiagonální systém rovnic

$$
\begin{array}{lr}
g_1x_1 + h_1x_2 & = b_1 \\
f_2x_1 + g_2x_2 + h_2x_3 & = b_2 \\
\quad\quad \cdots\cdots\cdots & \\
\quad\quad\quad f_ix_{i-1} + g_ix_i + h_ix_{i+1} & = b_i \\
\quad\quad\quad\quad\quad \cdots\cdots\cdots & \\
\quad\quad\quad\quad\quad\quad f_nx_{n-1} + g_nx_n & = b_n
\end{array}
$$

Hlavní trik algoritmu: přepiš i-tou rovnici do následujícího rekurentního tvaru

$$
\begin{bmatrix} x_{i+1} \\ x_i \\ 1 \end{bmatrix} = \mathcal{G}_i \begin{bmatrix} x_i \\ x_{i-1} \\ 1 \end{bmatrix},
$$

$$
\text{kde} \qquad \mathcal{G}_i = \begin{bmatrix} -\frac{g_i}{h_i} & -\frac{f_i}{h_i} & \frac{b_i}{h_i} \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad 1 \le i \le n-1.
$$

Opakovanou substitucí dostaneme

$$
\begin{bmatrix} x_{i+1} \\ x_i \\ 1 \end{bmatrix} = \mathcal{H}_i \begin{bmatrix} x_1 \\ 0 \\ 1 \end{bmatrix}, \quad \text{kde} \qquad \mathcal{H}_i = \mathcal{G}_i\mathcal{G}_{i-1}\ldots\mathcal{G}_1, \qquad 1 \le i \le n-1. \tag{1}
$$

Tridiagonální systém rovnic (pokr.)

Algoritmus na p procesorech $P_0, \ldots, P_{p-1}$ pracuje takto:

1. Aplikováním PPS na pole matic $\mathcal{G}_1, \ldots, \mathcal{G}_{n-1}$ procesory vypočtou $\mathcal{H}_1, \ldots, \mathcal{H}_{n-1}$. Asociativním operátorem je <u>násobení</u> (3×3)-matic ($O(n/p + \log p)$ kroků).

2. Procesor P_{p-1} vyřeší 3 rovnice

$$\begin{bmatrix} x_n \\ x_{n-1} \\ 1 \end{bmatrix} = \mathcal{H}_{n-1} \begin{bmatrix} x_1 \\ 0 \\ 1 \end{bmatrix},$$

$$f_n x_{n-1} + g_n x_n = b_n$$

   s 3 neznámými a vypočte x_1 ($O(1)$ kroků).

3. P_{p-1} vyšle hodnotu x_1 všem procesorům ($O(\log p)$ kroků).

4. Všechny procesory paralelně vypočtou x_{i+1} ze všech svých rovnic (1) ($O(n/p)$ kroků).

Segmentovaný paralelní prefixový součet (SPPS)

- Vstupní pole je rozděleno do segmentů.
- Cílem je vypočítat všechny prefixové součty uvnitř těchto segmentů izolovaně.

Příklad: 4-segmentové vstupní pole:

$$\mid 2 \quad 1 \quad 3 \quad 5 \mid 2 \quad 7 \quad 3 \mid 9 \quad 4 \quad 5 \quad 6 \mid 2 \quad 8 \quad 4 \quad 3 \quad 1 \mid$$

Provedením 4 izolovaných paralelních prefixových součtů s operací + získáme

$$\mid 2 \quad 3 \quad 6 \quad 11 \mid 2 \quad 9 \quad 12 \mid 9 \quad 13 \quad 18 \quad 24 \mid 2 \quad 10 \quad 14 \quad 17 \quad 18 \mid$$

♣

Hlavní myšlenka SPPS

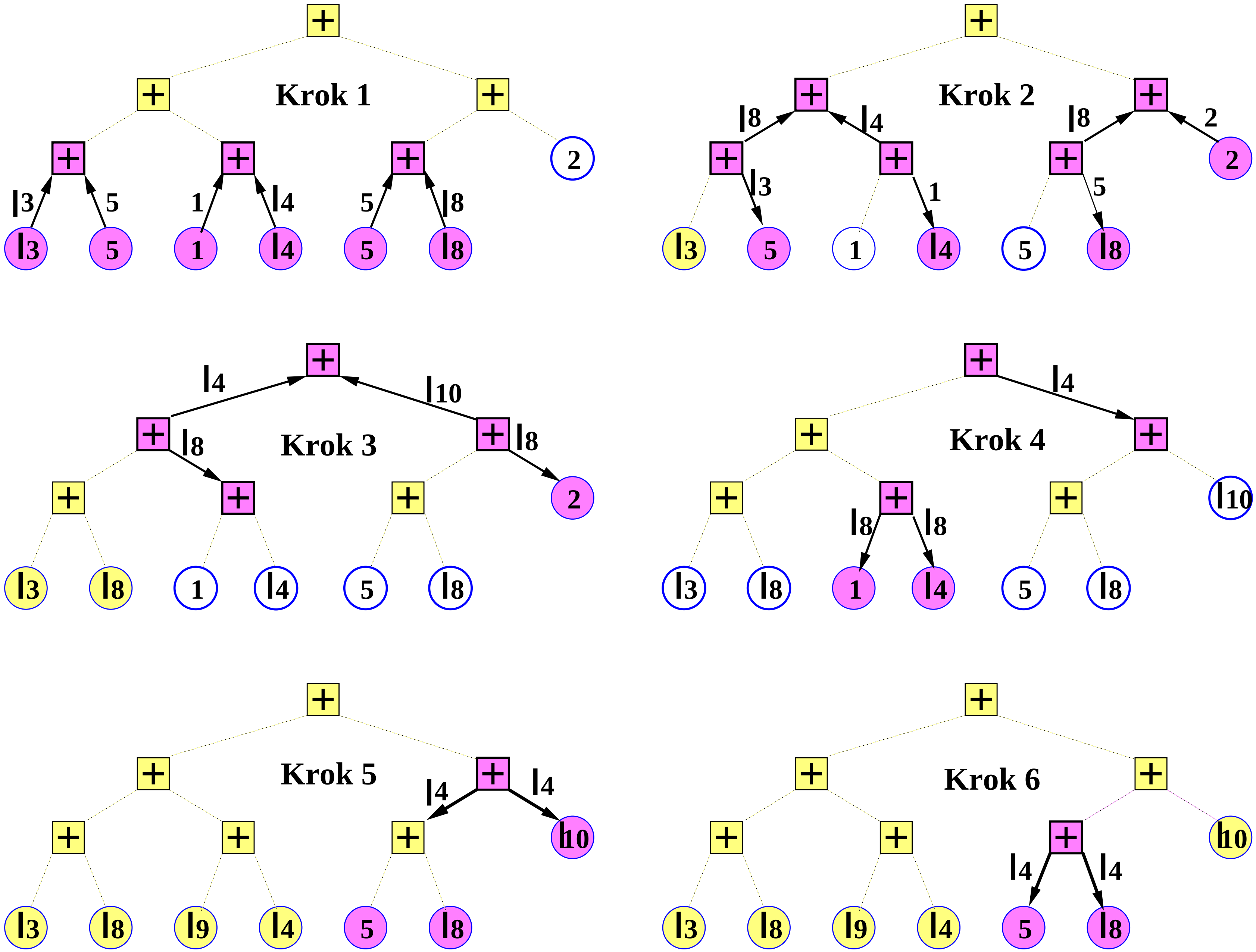
SPPS = 1 globální PPS aplikovaný na celé pole s modifikovanou bin. operací $\overline{\oplus}$

$\overline{\oplus}$	b	$\mid b$
a	$a \oplus b$	$\mid b$
$\mid a$	$\mid(a \oplus b)$	$\mid b$

Lemma 6. *Je-li $\oplus$ asociativní, je asociativní i $\overline{\oplus}$.* ♣

Příklad segmentovaného paralelního prefixového součtu

Vstupní pole: | 3 5 1 | 4 5 | 8 2 |



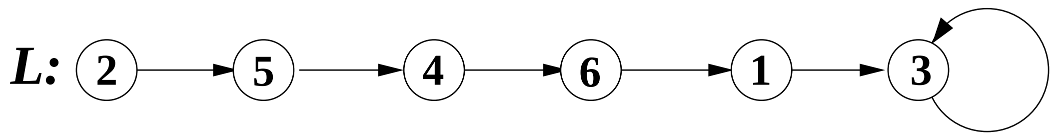
EREW PRAM paralelní QuickSort

- Jeden z několika paralelních QuickSort algoritmů.
- Vstupní posloupnost $A = \{a_0, a_2, \ldots, a_{n-1}\}$ je rozdělena rovnoměrně mezi $p < n$ procesorů.
- Z počátku, $A =$ jeden globální segment $S_0 = |a_0, a_2, \ldots, a_{n-1}$.
- V jedné iteraci, každý segment S je rozdělen do 3 podsegmentů $S_<$, $S_=$, $S_>$.
- Každá iterace vyžaduje $O(1)$ volání SPPS a $O(1)$ paralelních PRAM operací.
- Je-li A náhodná, pak by QuickSort měl skončit v $O(\log n)$ iteracích
  $\implies \quad T_{\text{QuickSort}}(n,p) = O(\log n) T_{\text{PPS}}(n,p)$.

B:**for** $i := 0, \ldots, n-2$ **do_in_parallel** $f_i := (a_i \leq a_{i+1})$; ($*$ je A setříděný? $*$)
proveď paralelní redukci hodnot f_i operací AND;
if (A je setříděný) **then EXIT**;
for všechny momentální segmenty S **v** A **do_in_parallel**
$\{$ vyber první prvek segmentu S jako jeho pivot b_S;
($*$ poslání hodnot všech pivotů přesně těm procesorům, které je potřebují $*$)
pomocí SPPS s $\oplus$ definovanou jako $a \oplus b = a$ pošli b_S procesorům uvnitř S;
$\forall\ a_i \in S$ **do_in_parallel**
$g_i := (a_i <> b_S)$, kde $g_i \in \{\texttt{'<'}, \texttt{'='}, \texttt{'>'}\}$;
($*$ nyní musíme každý S rozdělit do 3 podsegmentů $S_<$, $S_=$, $S_>$ $*$)
pomocí SPPS výpočti nové indexy prvků $\{a_i; g_i = \texttt{'<'}\}$ uvnitř všech S;
pošli hodnoty maximálního indexu ($= |S_<|$) uvnitř všech S (SPPS doleva);
($*$ $|S_<|$ je rovno indexu začátku $S_=$ $*$)
($*$ první prvek s indexem $1 =$ okrajový prvek $S_<$ $*$)
pomocí SPPS výpočti nové indexy prvků $\{a_i; g_i = \texttt{'='}\}$ uvnitř všech S;
pošli hodnoty maximálního indexu ($= |S_=|$) uvnitř všech S (SPPS doleva);
($*$ $|S_<| + |S_=|$ je rovno indexu začátku $S_>$ $*$)
($*$ první prvek s indexem $1 =$ okrajový prvek $S_=$ $*$)
pomocí SPPS výpočti nové indexy prvků $\{a_i; g_i = \texttt{'>'}\}$ uvnitř všech S;
($*$ první prvek s indexem $1 =$ okrajový prvek $S_>$ $*$)
permutací segmentu S vytvoř 3 podsegmenty $S = \{S_<|S_=|S_>\}$;
$\}$;
goto B;

Zřetězené seznamy

- n-prvkový jednosměrně zřetězený seznam L je reprezentován pomocí pole následníků S (successors).
- Každý prvek má jednoznačné ID z $\{1, \ldots, n\}$.



S:

1	2	3	4	5	6
3	5	3	6	4	1

P:

1	2	3	4	5	6
6	2	1	5	2	4

A:

1	2	3	4	5	6
2	5	4	6	1	3

Konverze jednosměrně zřetězeného seznamu na obousměrně

- Platí: $P[i] = j \quad \Longleftrightarrow \quad S[j] = i$.
- Plně paralelní (t.j. lineárně škálovatelný) EREW PRAM(n, p) algoritmus: $T(n, p) = O(n/p)$.

Algorithm EREW_PRAM_SINGLE2DOUBLE (**in:** $S[1, \ldots, n]$; **out:** $P[1, \ldots, n]$)

for all $i := 1, \ldots, n$ **do_in_parallel**
  $\{P[i] := i$; **if** $(S[i] \neq i)$ **then** $P[S[i]] := i\}$

Výpočet pořadí v zřetězeném seznamu

- Pořadí = vzdálenost (= počet ukazatelů) do konce seznamu.
- Sekvenční řešení: triviální algoritmus projitím ukazatelů.

Výpočet pořadí pomocí přeskoku ukazatelů (PU)

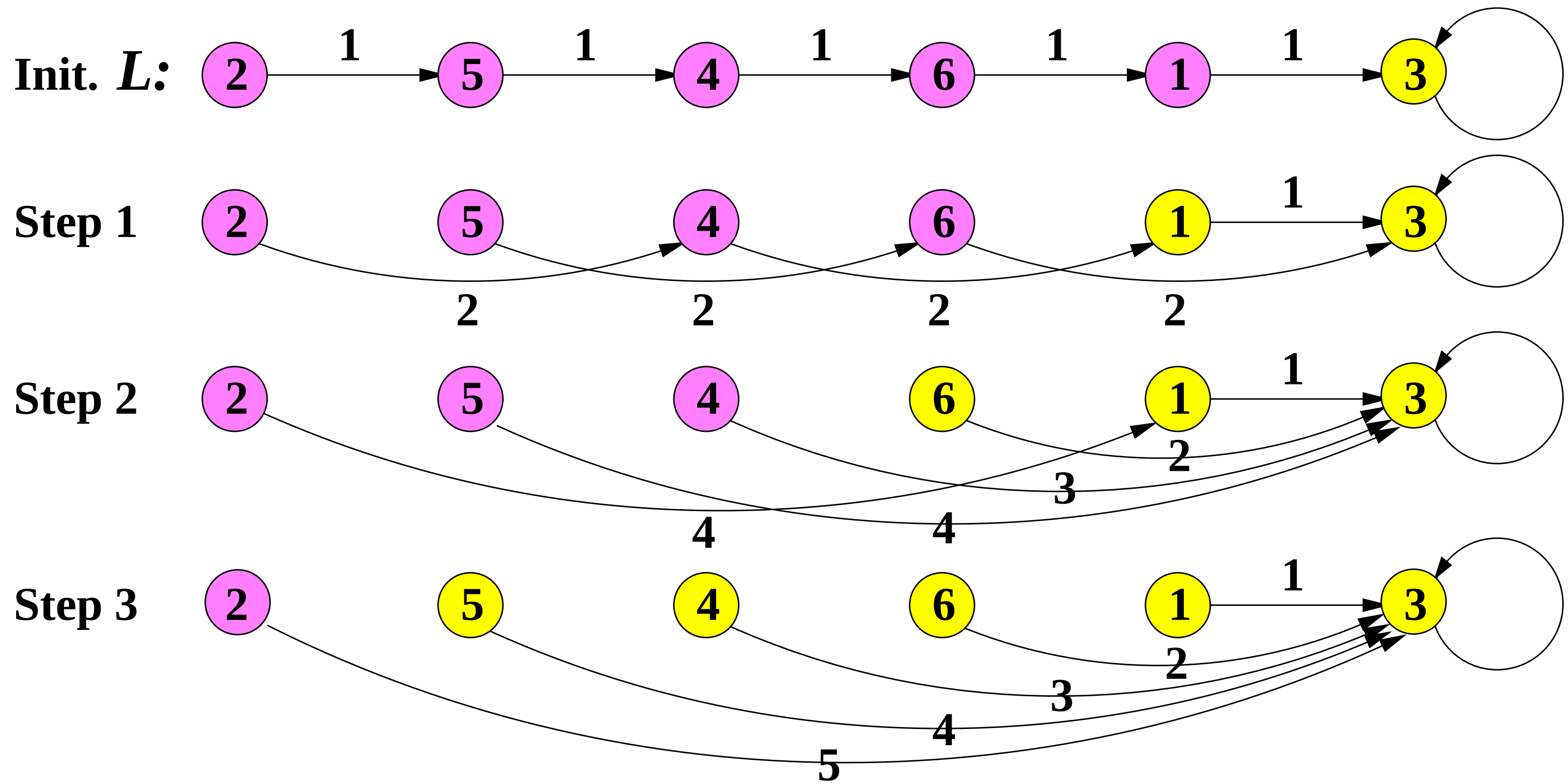
Algorithm $\textsc{CREW\_PRAM\_List\_Ranking}$ (**in,out**: $S[1,\ldots,n]$, **out:** $R[1,\ldots,n]$)

for all $i := 1,\ldots,n$ **do_in_parallel**
  { **if** $(S[i] = i)$ **then** $R[i] := 0$ **else** $R[i] := 1$;
   repeat $\lceil \log n \rceil$ **times**
     { $R[i] := R[i] + R[S[i]]$; $S[i] := S[S[i]]$}} (∗ pointer jumping ∗)

Paralelní postfixový součet na zřetězeném seznamu

Alg. $\textsc{CREW\_PRAM\_Parallel\_Suffix\_Sum}$ (**in,out**: $S[1,\ldots,n]$, **out:** $V[1,\ldots,n]$)

for all $i := 1,\ldots,n$ **do_in_parallel**
  { **if** $(S[i] = i)$ **then** $V[i] := 0$ **else** $V[i] := v_i$;
   repeat $\lceil \log n \rceil$ **times**
     { $V[i] := V[i] \oplus V[S[i]]$; $S[i] := S[S[i]]$ }; (∗ pointer jumping ∗)
   if (original $V[last] \neq 0$) **then** $V[i] := V[i] \oplus V[last]$}



	S: 1	2	3	4	5	6
Init.	3	5	3	6	4	1
Step 1	3	4	3	1	6	3
Step 2	3	1	3	3	3	3
Step 3	3	3	3	3	3	3

	R: 1	2	3	4	5	6
Init.	1	1	0	1	1	1
Step 1	1	2	0	2	2	2
Step 2	1	4	0	3	4	2
Step 3	1	5	0	3	4	2

Důležitost výpočtu pořadí prvků v seznamu

- Znalost pořadí dovoluje transformovat zřetězený seznam na lineární pole: prvky jsou permutovány na pozice dané svým pořadím v seznamu.
- Na tomto novém poli lze po té veškeré další výpočty provádět pomocí PPS.

Škálovatelnost přeskakování ukazatelů (PU)

- CREW PRAM(n, n) PU není cenově optimální: $C(n, n) = O(n \log n)$.
- PU je datově necitlivé: provádí přeskoky i přes ukazatele na poslední prvek. Tudíž taktéž $W(n, n) = O(n \log n)$.
- Avšak, cenovou optimalizaci nelze založit na zmenšení p a přidělení n/p prvků každému procesoru, neboť
  - Po kroku i je # spočítaných prvků $2^{i-1} + 1$ pro $i \geq 1$ (viz žluté prvky x na předchozím obr.).
  - $\Theta(n/2)$ prvků vyžadují $\Theta(\log n)$ přeskoků, než se dostanou na konec.
  - CREW PRAM(n, p) algoritmus PU může být pracovně-optimální pouze, když každý P udržuje seznam aktivních (=nedopočítaných) prvků.
  - Ale ani tato optimalizovaná verze nezlepší cenu, protože 1 P může mít n/p aktivních prvků v každém z $\log n$ kroků a tudíž, $T(n, p) = O((n/p) \log n)$ a $C(n, p) = O(n \log n)$.

Pozorování

Uvažujme zřetězený seznam s $n' = n/\log n$ prvky
a
použijme přeskakování ukazatelů pomocí $p = n/\log n$ procesorů
$\implies \quad T(n', p) = O(\log n)$ a $C(n', p) = O(n)$.

Myšlenka škálovatelného algoritmu pro výpočet pořadí v seznamu

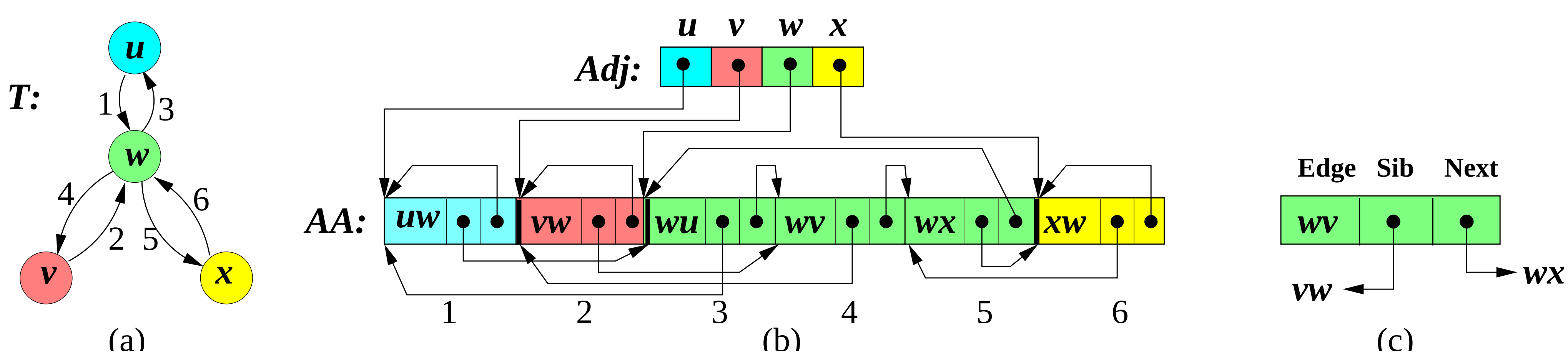
0. Nechť $|L| = n$ prvků a $p = \Theta(n/\log n)$.
1. Transformuj L na L' velikosti $n' = O(n/\log n)$ s $C_1(n, p) = O(n)$,
   t.j., v $T_1(n, p) = O(\log n)$ pomocí p procesorů.
2. Proveď přeskakování ukazatelů na L'. Pak
   $T_2(n', p) = O(\log n') = O(\log n)$ a $C_2(n', p) = O(n)$.
3. Restauruj L z L' a vypočti konečné pořadí pro prvky v $L - L'$
   s toutéž složitostí jako v kroku 2, tedy $T_3(n, p) = O(T_1(n, p))$.

Eulerovské cesty a Eulerovské grafy

- Eulerovské cykly G = kružnice, které procházejí každou hranou G přesně jednou.
- G je eulerovský, má-li každý uzel G sudý stupeň.
- Neorientovaný strom může být transformován na eulerovský, je-li každá hrana nahrazena dvojicí anti-paralelních hran. (a)

Reprezentace grafu a Eulerovské stromy

- Pole uzlů (Adj): $x \in V(T)$, $Adj[x]$ = ukazatel do pole hran AA. (b)
- Pole hran (AA) se skládá z podseznamů hran incidentních s jednotlivými uzly. (b)+(c)



Hrany $\langle x \rightarrow y \rangle$ jsou pro jednoduchost značeny xy. Sib je ukazatel na anti-paralelní dvojče.

Paralelní konstrukce eulerovských cest a kružnic

Eulerovská kružnice stromu $T' =$ cyklický zřetězený seznam hran, reprezentovaný jako pole následníků ET:

ET:

4	5	1	2	6	3
1	2	3	4	5	6

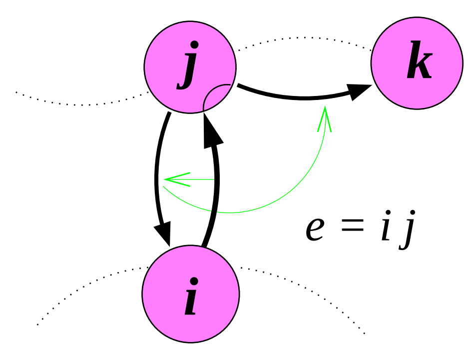
Eulerovská kružnice může být vypočítána plně paralelním EREW PRAM algoritmem.

Algorithm EREW_PRAM_EULERIAN_TOUR (**in:** AA; **out:** ET;)

for all arcs e in AA of a tree T **do_in_parallel**

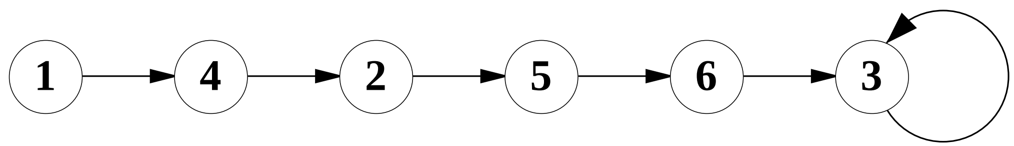
$ET[e] := (AA[e].Sib\uparrow).Next$

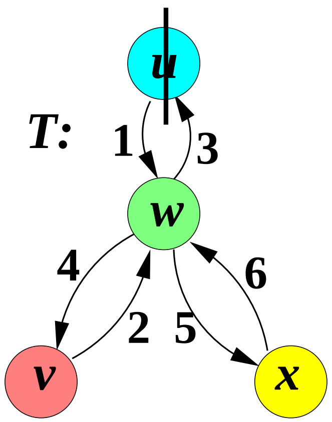
Pravidlo procházení bludištěm:
kráčej kupředu a stále přidržuj pravou ruku v kontaktu s pravou zdí.



Eulerovská cesta

Rozetni eulerovskou kružnici ve zvoleném uzlu.





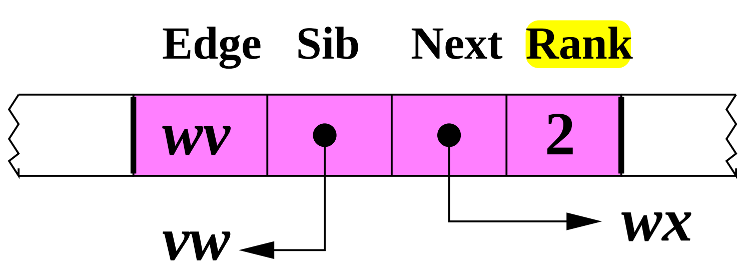
EP:

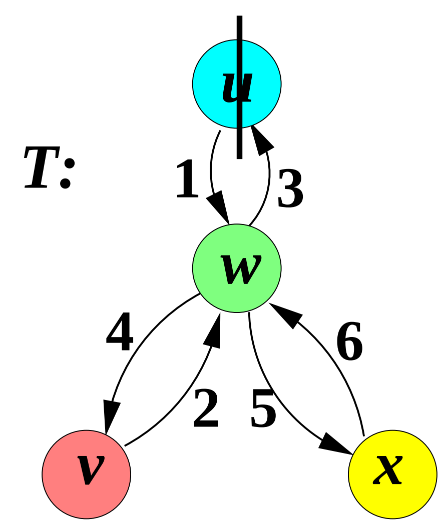
4	**5**	**3**	**2**	**6**	**3**
1	**2**	**3**	**4**	**5**	**6**

Výpočet pořadí

Pořadí hrany v $T' =$ její pozice od začátku eulerovské cesty (pole $Rank[\ldots]$)
Pak EP (zřetězený seznam) může být paralelně transformován na pole EA.

AA:





EA:

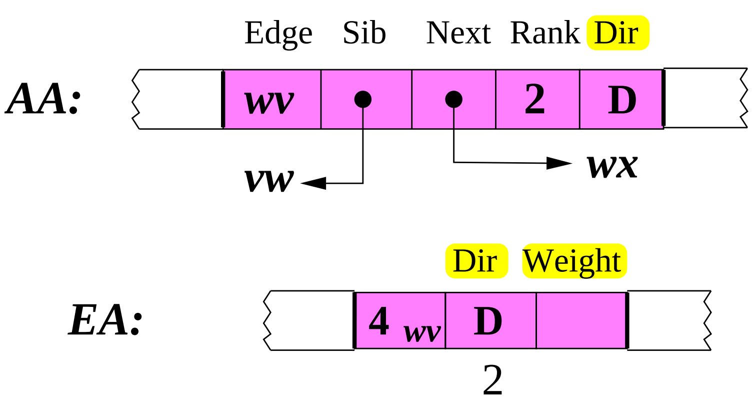
1 uw	**4** wv	**2** vw	**5** wx	**6** xw	**3** wu
1	2	3	4	5	6

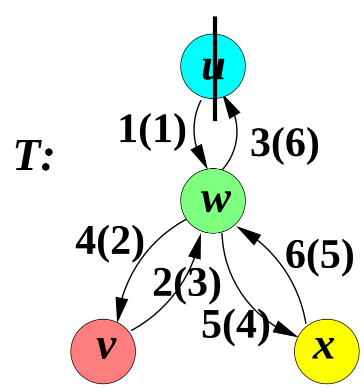
Kořenový strom = nalezení všech rodičů paralelně

Myšlenka: hrany v eulerovské cestě označ jako jdoucí <u>dolu</u> (D = down) nebo <u>nahoru</u> (U = up).

Alg. EREW_PRAM_ALLPARENTS (**In:** $AA, EA[1, \ldots, m]$; **Out:** $Parent[1, \ldots, n]$);

for all arcs xy **do_in_parallel**
  if $(AA[xy].Rank < AA[yx].Rank)$ **then**
    $\{\ AA[xy].Dir := EA[AA[xy].Rank].Dir :=$ `Down`;
    $AA[yx].Dir := EA[AA[yx].Rank].Dir :=$ `Up`; $Parent[y] := x;\ \}$;
$Parent[root] :=$ `nil`;





Parent:

u	v	w	x
/	w	u	w

Výpočet velikostí všech podstromů paralelně I

Myšlenka: počet uzlů v podstromu $= 1/2$ počtu hran podstromu.

Alg. $\mathrm{EREW\_PRAM\_A{\scriptsize LL}S{\scriptsize UBTREE}S{\scriptsize IZES}I}$ (**In:** AA, $Parent$; **Out:** $Subtree[1,..,n]$);

for all vertices $i \neq root$ in T **do_in_parallel**

$Subtree[i] = (AA[ij].Rank - AA[ji].Rank + 1)/2$, where $j = Parent[i]$;

$Subtree[root] := n$;

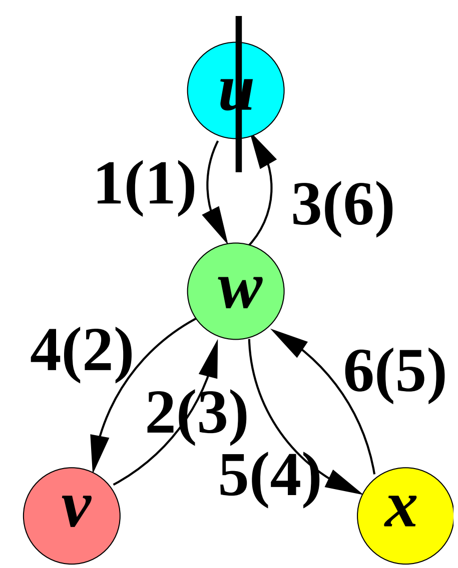
Parent:

u	***v***	***w***	***x***
	w	***u***	***w***

Subtree:

u	***v***	***w***	***x***
4	**1**	**3**	**1**

T:



Výpočet velikostí všech podstromů paralelně II

Myšlenka: velikost podstromu = počet jeho `Down` hran.

Alg. EREW_PRAM_ALLSUBTREESIZESII (**In:** $EA[1,..,m]$, **Out:** $Subtree[1,..,n]$);

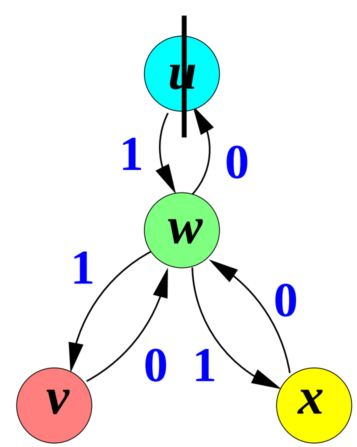
for all arcs $e \in EA$ **do_in_parallel**
    if $(EA[e].Dir = \texttt{Down})$ **then** $EA[e].Weight := 1$ **else** $EA[e].Weight := 0$;
apply PSS on the array $EA[1,...,m].Weight$;
for all arcs $xy \in EA$ **do_in_parallel**
    if $(EA[xy].Dir = \texttt{Down})$ **then** $Subtree[y] := EA[xy].Weight - EA[yx].Weight$;
$Subtree[root] := n$;

EA:

uw	D	1	*wv*	D	1	*vw*	U	0	*wx*	D	1	*xw*	U	0	*wu*	U	0

uw	D	3	*wv*	D	2	*vw*	U	1	*wx*	D	1	*xw*	U	0	*wu*	U	0

T:



Výpočet úrovní všech uzlů paralelně

Myšlenka: paralelní párování závorek.

Alg. $\text{EREW\_PRAM\_AllLevels}$ (**In:** $EA[1,\ldots,m]$, **Out:** $Level[1,\ldots,n]$);

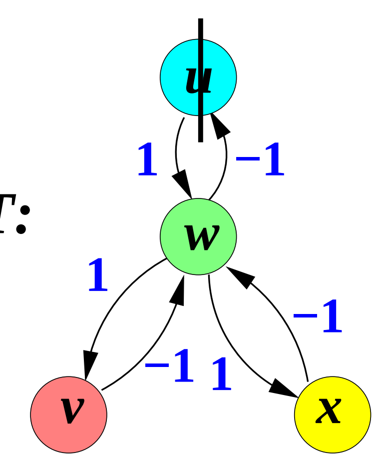
for all arcs $e \in EA$ **do_in_parallel**
  if $(EA[e].Dir = \texttt{Down})$ **then** $EA[e].Weight := 1$ **else** $EA[e].Weight := -1$;
apply PPS on array $EA[1,\ldots,m].Weight$;
for all arcs $xy \in EA$ **do_in_parallel**
  if $(EA[xy].Dir = \texttt{Down})$ **then** $Level[y] := EA[xy].Weight$;
$Level[root] := 0$;

EA:

uw	D	1	*wv*	D	1	*vw*	U	−1	*wx*	D	1	*xw*	U	−1	*wu*	U	−1

uw	D	1	*wv*	D	2	*vw*	U	1	*wx*	D	2	*xw*	U	1	*wu*	U	0

T:



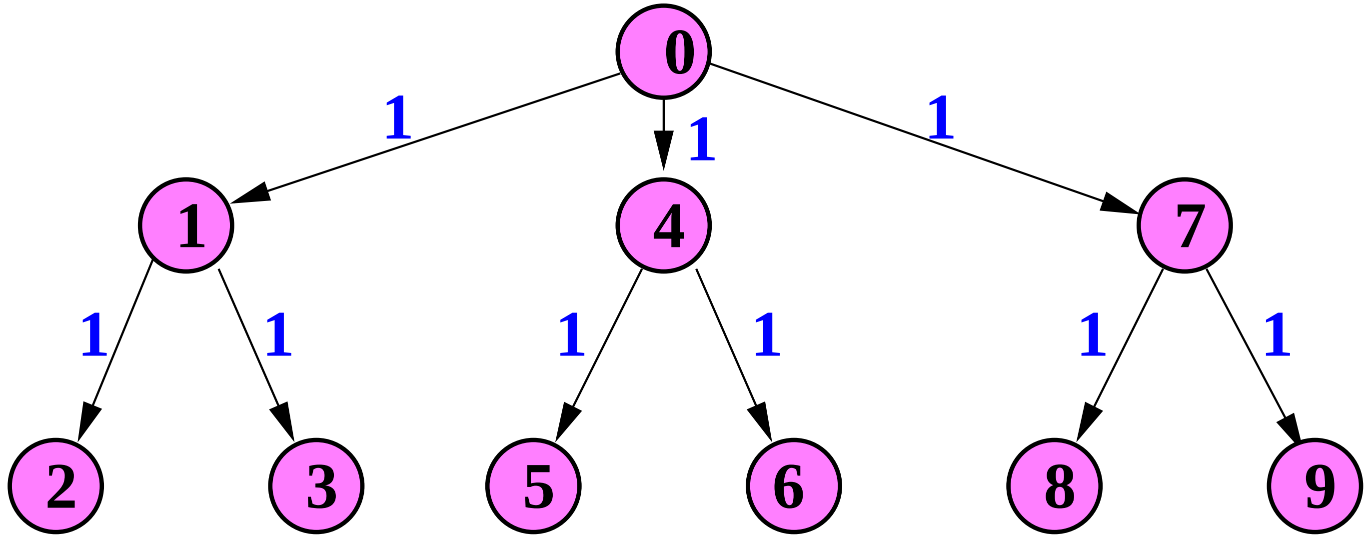
Číslování Preorder

- průchod Preorder (počínaje od 0): kořen, levý podstrom, pravý podstrom.

Myšlenka: Preorder pořadí uzlu = počet `Down` hran v ET z kořene do uzlu.

Alg. $\mathrm{EREW\_PRAM\_P{\scriptsize RE}O{\scriptsize RDER}}$ (**In:** $EA[1,\ldots,m]$; **Out:** $Pre[1,\ldots,n]$);

for all arcs $e \in EA$ **do_in_parallel**
  if $(EA[e].Dir = \texttt{Down})$ **then** $EA[e].Weight := 1$ **else** $EA[e].Weight := 0$;
apply PPS on $EA[1,\ldots,m].Weight$;
for all arcs $xy \in EA$ **do_in_parallel**
  if $(EA[e].Dir = \texttt{Down})$ **then** $Pre[y] := EA[xy].Weight$;



Závěr

Technika eulerovské cesty je základem pro široké spektrum dalších paralelních výpočtů na grafech, jako např.

- výpočet minimální kostry,
- vrcholová/hranová souvislost,
- ear decomposition search,
- testování planarity.