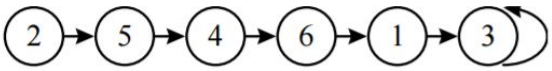
31. Distribuované a paralelní algoritmy - algoritmy nad seznamy, stromy a grafy

31. Distribuované a paralelní algoritmy - algoritmy nad seznamy, stromy a grafy

// přednáška PRL č. 7: https://www.fit.vutbr.cz/study/courses/PDA/private/www/h007.pdf

Práce se seznamy



Pole následníků, pole předchůdců

Výpočet předchůdce

Analýza

- t(n) = O(c)
- p(n) = n
- c(n) = O(n)

Algoritmus

```
for i = 1 to n do in parallel
     if i<>Succ[i] then //ošetření prvního prvku
     Pred[Succ[i]] = i;
```

Parallel suffix sum

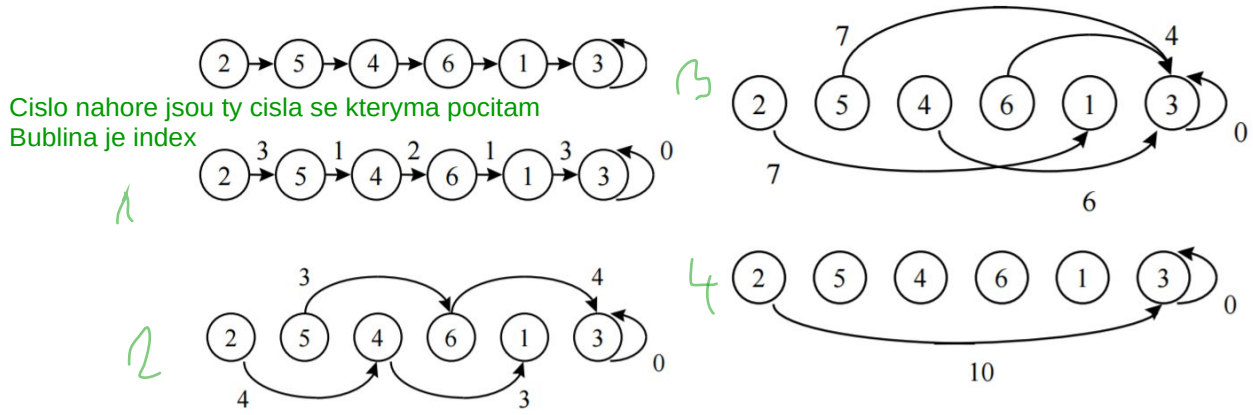
- Vypočítává součet suffixů (podobně jako suma prefixů) seznamu
- **Suffix** – podseznam mezi prvkem a koncem seznamu

Analýza

- t(n) = O(log n)
- c(n) = O(n.log n)

Algoritmus

```
V = [vn-1, ... v1,0] // pole hodnot
for i = 1 to n do in parallel
if Succ[i] = i then
Val[i] = 0 /* neboli neutrální prvek operace ⊕ */
else Val[i] = vi
for k = 1 to log n do
Val[i] = Val[i] ⊕ Val[Succ[i]]
Succ[i] = Succ[Succ[i]]
end for
if Val[last] <> 0 then
Val[i] = Val[i] ⊕ Val[last]
end for
```



List ranking

Princip totozny jako predchozi algoritmus, pouze jsou misto hodnot uzlu vsude hodnoty 1

List ranking

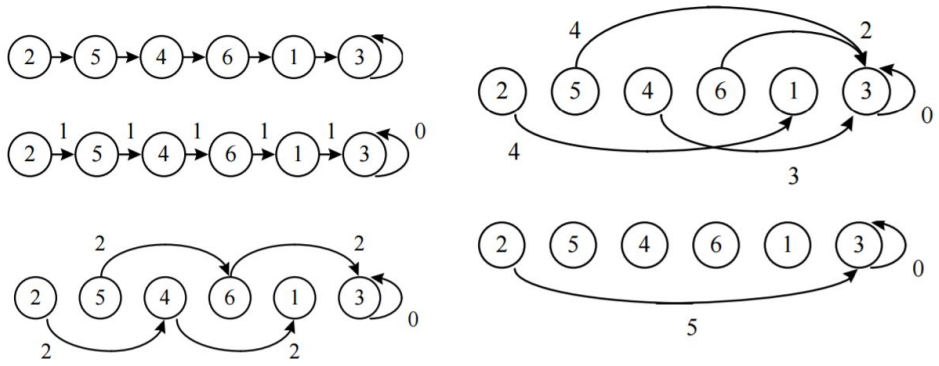
- Nalezni pořadí (rank) prvků seznamu (vzdálenost od konce seznamu)
- Sekvenční algoritmus má složitost O(n)

Analýza

- t(n) = O(log n)
- p(n) = n
- c(n) = O(n.log n) – není optimální

Algoritmus

```
Input: array Succ[1..n]
Output: array Rank[1..n]
for i=1 to n do in parallel
     if Succ[i]=i then
          Rank[i] = 0   Posledni prvek si da 0, ostatni jedna
          else Rank[i] = 1
for k = 1 to log n do
          Rank[i] = Rank[i] + Rank[Succ[i]]
          Succ[i] = Succ[Succ[i]]
     end for
end for
```

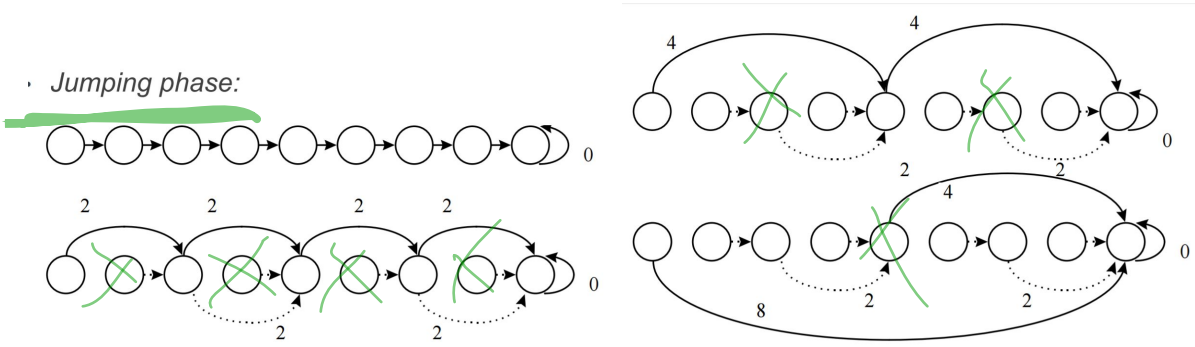


Byl problem, ze procesory co se preskocili byly nevyuzite po zbytek vypoctu - resenim je, ze je uspim

List ranking revisited

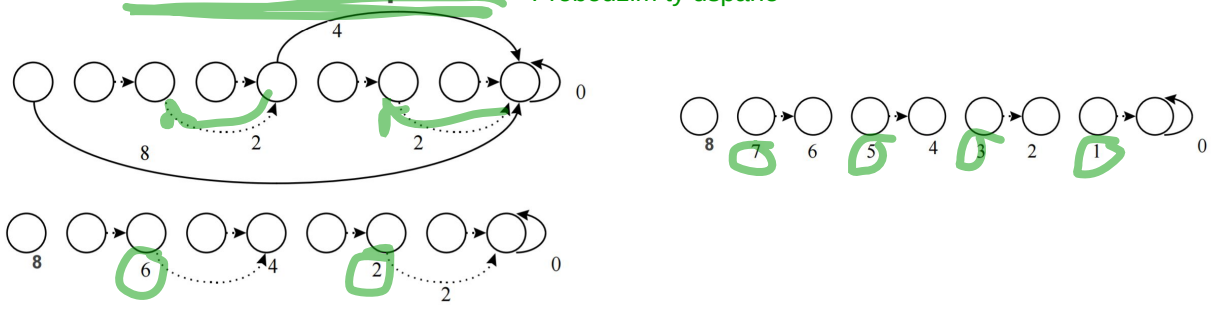
- List ranking neefektivní - zbytečně se opakuje výpočet
- Řešení: v každém kole přestane pracovat polovina procesorů

- *Jumping phase:*



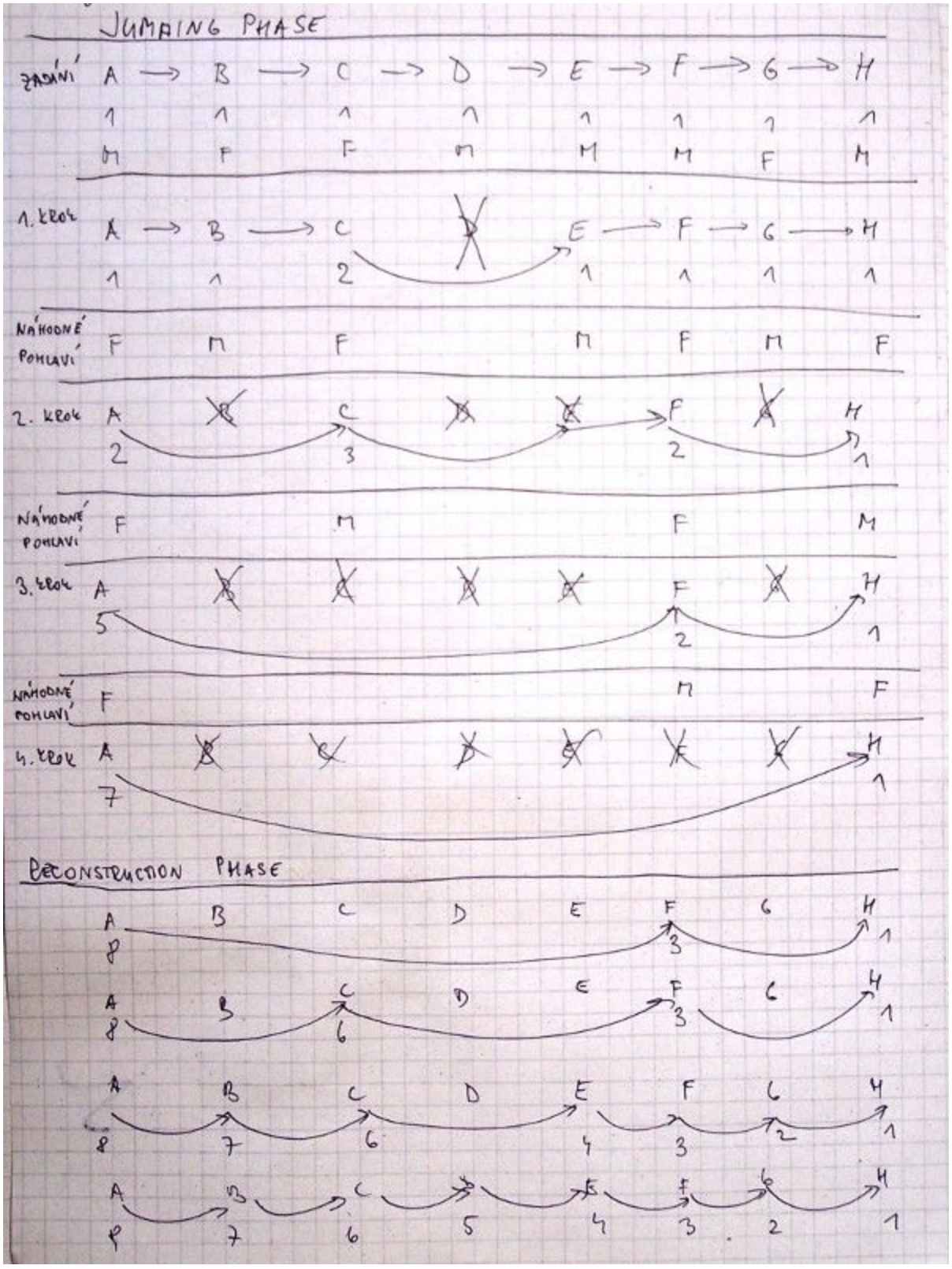
Reconstruction phase

Probouzim ty uspane



Random mating

- Optimalizovaný algoritmus list ranking
- Každý procesor si hodí korunou a přiřadí si pohlaví (male/female)
- Pokud je procesor female a jeho následník je male, pak se prvek male přeskočí (jump over) a procesor se uvolní

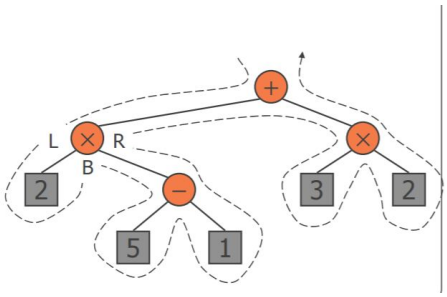

JUMPING PHASE
ZADÁNÍ
A → B → C → D → E → F → G → H
1 1 1 1 1 1 1 1
M F F M M M F M
1. KROK
A → B → C E → F → G → H
1 1 2 1 1 1 1
NÁHODNÉ POHLAVÍ
F M F M F M F
2. KROK
A C F H
2 3 2 1
NÁHODNÉ POHLAVÍ
F M F M
3. KROK
A F H
5 2 1
NÁHODNÉ POHLAVÍ
F M F
4. KROK
A H
7 1
DECONSTRUCTION PHASE
A B C D E F G H
8 3 1
A B C D E F G H
8 6 3 1
A B C D E F G H
8 7 6 4 3 2 1
A B C D E F G H
8 7 6 5 4 3 2 1

Optimal list ranking

- Požadavek: pevný počet stále pracujících procesorů
- Simulace algoritmu Random Mate pomocí n/log n procesorů, každý procesor vykonává práci log n procesorů
- Každý procesor má zásobník prvků
- Každý procesor se pokouší přeskočit (jump over) následníka prvku na vrcholu zásobníku
- Pokud je vrchol zásobníku přeskočen, procesor se zabývá dalším prvkem na zásobníku
- Nevýhoda: algoritmus může být nevyvážený
- Řešení: místo jump over se použije splice out (vypletení)

Stromy

Euler tour traversal



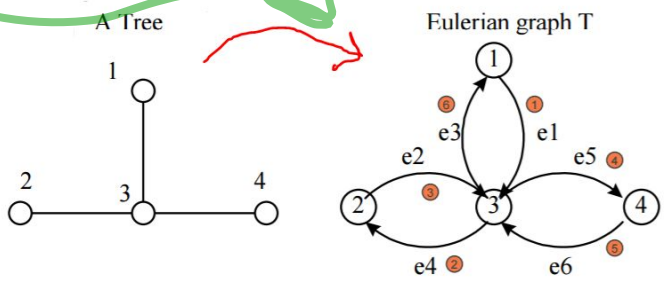
- Obecný průchod binárním stromem
- Její speciální případy jsou průchody preorder, postorder a inorder

Euler tour

- T = (V, E) - daný strom
- T' = (V, E') - orientovaný graf získaný z T tak, že každá hrana (u, v) je nahrazena dvěma orientovanými hranami <u, v> a <v, u>

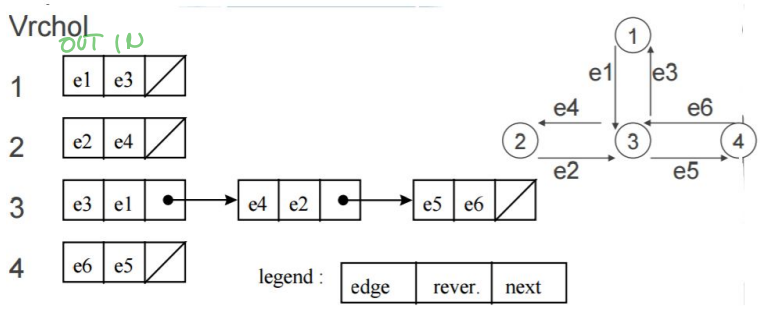
Eulerovský graf

- T' je Eulerovský graf – obsahuje orientovanou kružnici, která prochází každou hranou právě jednou



Eulerova kružnice

- Eulerova kružnice je reprezentována funkcí následníka Etour která každé hraně e∈ E' přiřazuje hranu Etour(e) ∈ E' která následuje hranu e
- Reprezentace – seznam sousednosti (adjacency list)
- Pokud e=(u, v) je hrana e, pak R = (v, u) je reverzní hrana

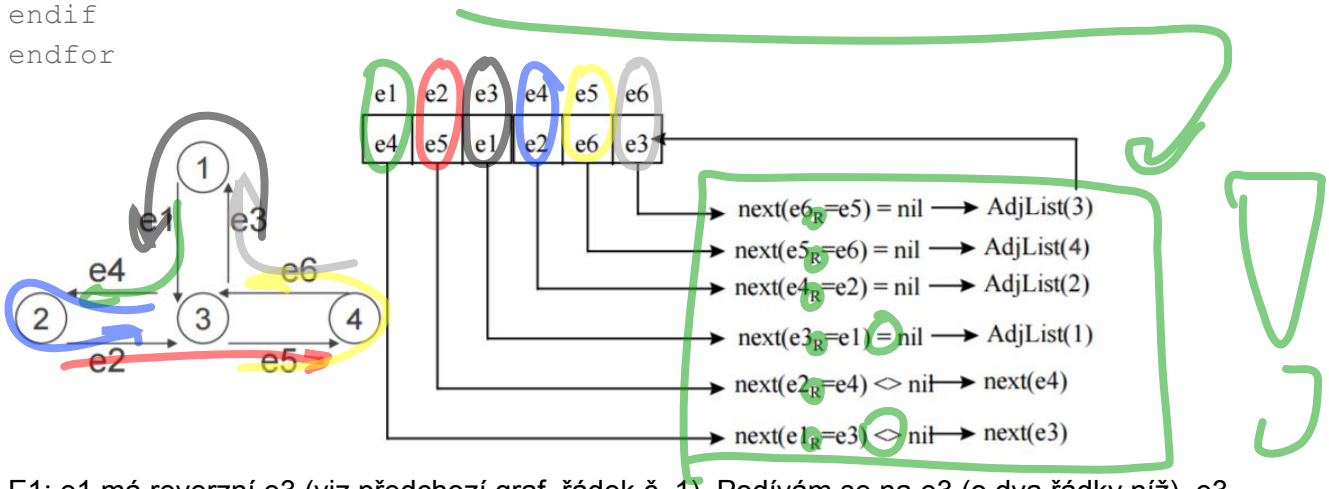


Vytvoření Eulerovy cesty

Algorithm

```
Input:     adjacency list of T
Output:    Array Etour with 2n-2 entries
Etour(e) is a edge following e
for i = 1 to 2n-2 do in parallel { ei = (u, v) }
if next(eR) <> nil then
Etour(e) = next(eR)
else Etour(e) = AdjList(v) { first item of adj. list of vertex v }
endif
endfor
```

Pokud ma naslednika tak naslednik
Jinak naslednik ciloveho vrcholu



E1: e1 má reverzní e3 (viz předchozí graf, řádek č. 1). Podívám se na e3 (o dva řádky níž). e3 má next e4, proto e4.
E2: e2 má následnou e4 (řádek 2). E4 (řádek 3) má následnou e5
E3: e3 má následnou e1 (řádek č. 3). E1 (řádek 1) nemá žádného následníka = je null.

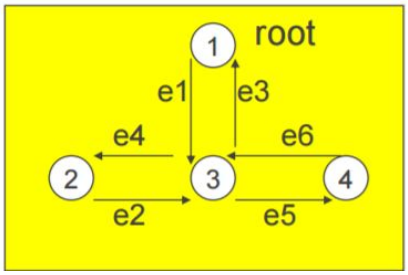
Zavedení kořene

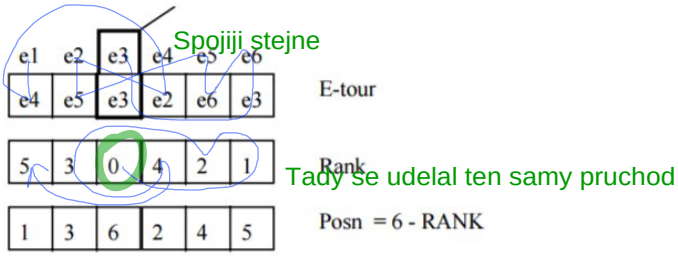
- Hrana vedoucí do kořene (př. e3)

Spočtení pozice každé hrany

- Algoritmus – Eulerova cesta s pozicemi –

- Vstup: binární strom (adjacency list) –
- Výstup: pole Etour, pole Posn, kde Posn(e) je pozice hrany e v Eulerově cestě
- 1. Spočteme se pole Etour O(c)
- 2. Přiřadíme Etour(e) = e pro hranu e vedoucí do kořene O(1)
- 3. Rank = ListRanking(Etour) O(log n)
- 4. Spočteme paralelně posn(e) = 2n-2-Rank(e) O(1) Reverzace poradi





Zacatek. Pak jdu podle e tour a pricitam jednicku

Obecný výpočet ve stromu

Pro mnoho algoritmů nad stromem T je postup:

- Vytvoříme Eulerovu cestu
- Vytvoříme pole hodnot
- Spočteme nad ním sumu suffixů
- Provedeme korekci

Můžeme tak např. spočíst:

- Pořadí postorder vrcholu
- Pořadí preorder vrcholu
- Pořadí inorder vrcholu
- Počet následníků vrcholu
- Úroveň vrcholu

Analýza

0) Spočtení Etour O(c)
1) Inicializace weight O(c)
2) Výpočet SuffixSums O(log n)
3) Korekce výsledku O(c)

- t(n) = O(log n)
- c(n) – závisí na implementaci SuffixSums

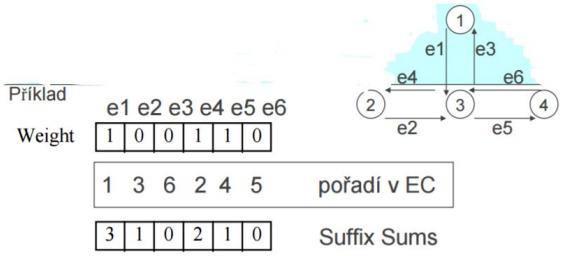
Přiřazení pořadí preorder vrcholům

- Pořadí preorder vrcholu ve stromě je 1+počet dopředných hran, kterými jsme prošli po cestě k vrcholu
- (Preorder – navštiv nejdřív otce, pak oba syny)

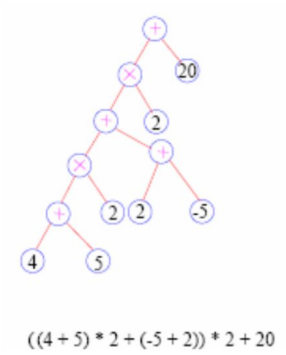
Algorithm

```
1) for each e do in parallel
if e is forward edge then
weight = 1
else weight = 0
endif
2) weight = SuffixSums(Etour,
Weight)
3) for each e do in parallel
if e=(u, v) is forward edge then
preorder(v) = n - weight(e) + 1
endif
   preorder(root) = 1
```

N je pocet vrcholu

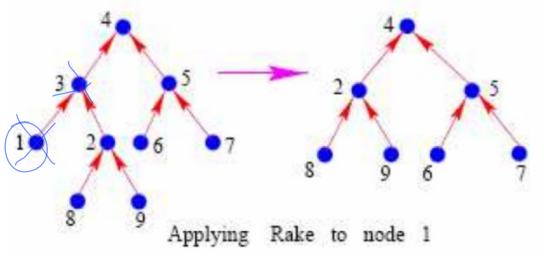


Tree Contraction



- Každý list obsahuje operand a každý nelist obsahuje operátor, jako je například +,∗
- Cílem je spočíst hodnotu výrazu v kořeni
- Technika tree contraction je systematický způsob jak zmenšovat strom až do velikosti jednoho vrcholu

The RAKE operation



Rake odstrani sebe a sveho otce

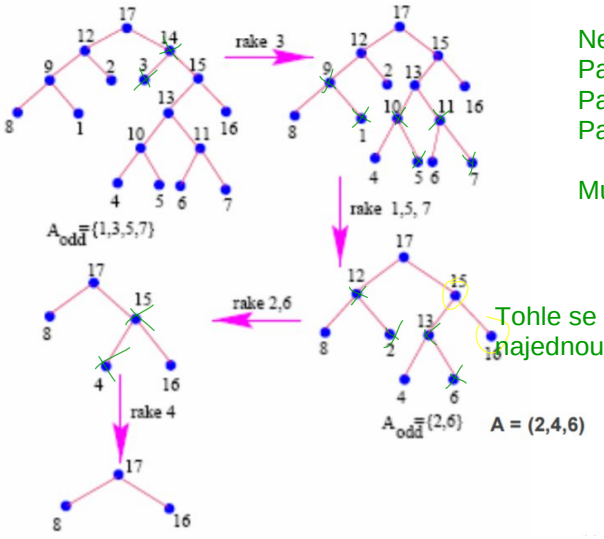
- Musíme aplikovat operaci RAKE na nesousedící uzly (např nemůžu zároveň provést RAKE na 8 a 1)

Algoritmus

- Označíme listy jejich pořadím zleva doprava

p(v) je parent uzlu v

- V Eulerově cestě se listy vyskytují také v pořadí zleva doprava
- Každé hraně (v, p(v)), kde v je listem, přiřadíme váhu 1
- Vyřadíme nejlevější a nejpravější list. Tyto listy budou dva synové kořene až se podaří strom zmenšit na strom se třemi vrcholy
- Nad výsledným seznamem provedeme sumu suffixů a získáme listy, očíslované zleva doprava



Nejdriv vsechny liche co jsou vlevo,
Pak vsechny liche ale v pravo.
Pak sude napravo
Pak sude nalevo

Musi to byt vzdycky list (a jeho otec)

Tohle se ted nemuze, protoze bych jakokdyby odstranil celou uroven najednou

- Provádíme RAKE na každého druhého levého syna, pak na každého druhého pravého syna atd.
- Počet listů se po každé iteraci cyklu zmenší na polovinu – Liché listy jsou „shrabány“. Proto je strom zcela zredukován v čase O(log n)