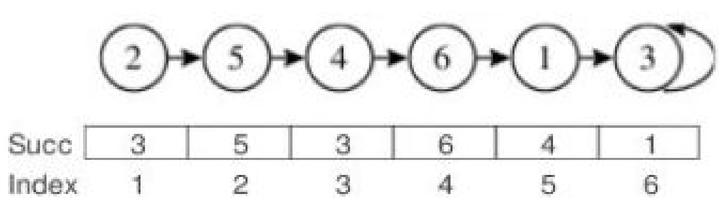
Distribuované a paralelní algoritmy - seznamy, stromy, grafy

Z FITwiki

Obsah

Seznamy

Lineární seznam
: pole prvků v paměti (možnost přistoupit indexem), které obsahují hodnotu v_i a ukazatel na následníka $succ[i]$. Poslední prvek ukazuje sám na sebe.



Predecessor computing

- počítá index předchůdce všech prvků
- t(n) = O(c)
- p(n) = n
- c(n) = O(n)

```
for i = 1 to n do in parallel
  Pred[Succ[i]] = i
```

Každý procesor vezme jeden prvek a následníkovi jeho prvku zapíše prvek jako předchůdce Pozn.: Je třeba zvlášť ošetřit první a poslední prvek seznamu

List ranking (sekvenční)

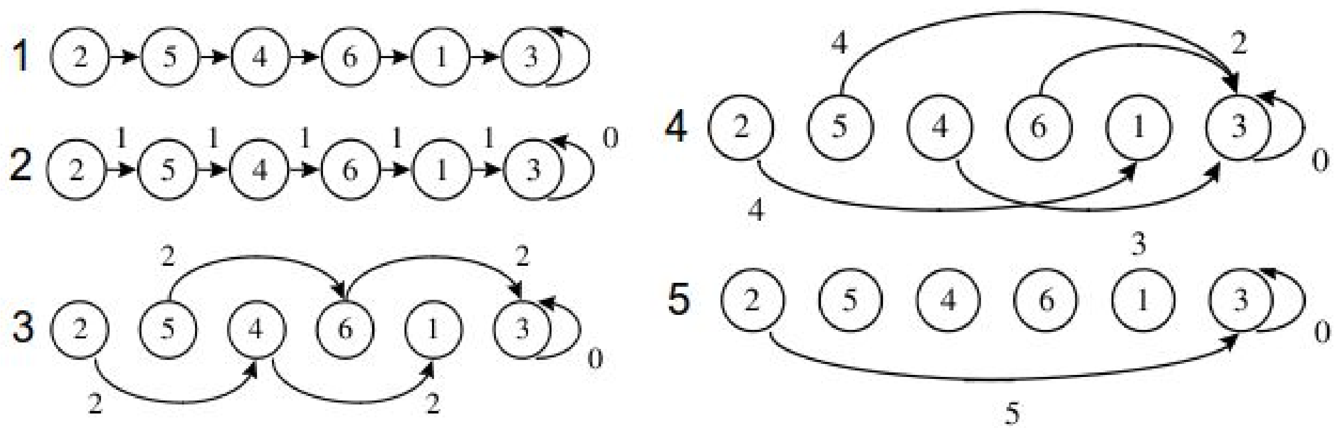
- přiřazuje prvkům rank = jejich vzdálenost od konce.
- Sekvenční časová složitost je $O(n)$
- V paralelním prostředí se používá technika **path doubling**.

List ranking (path doubling)

(Wyllie's algorithm)

```
Algorithm
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
                 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
```

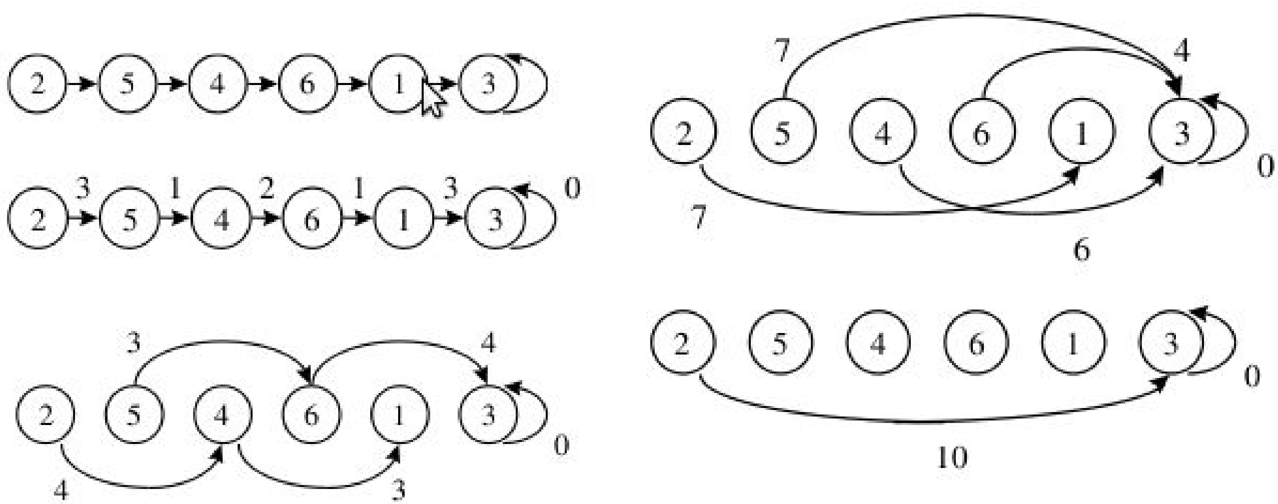
- 1) Paralelně se všem prvkům přiřadí RANK (0 pro poslední prvek, jinak 1).
- 2) Každý procesor prvku v $log(n)$ krocích
  - se počítá RANK jako `RANK[i] + RANK[Succ[i]]`
  - posune ukazatel `Succ[i] = Succ[Succ[i]]`



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

Paralení suma suffixů

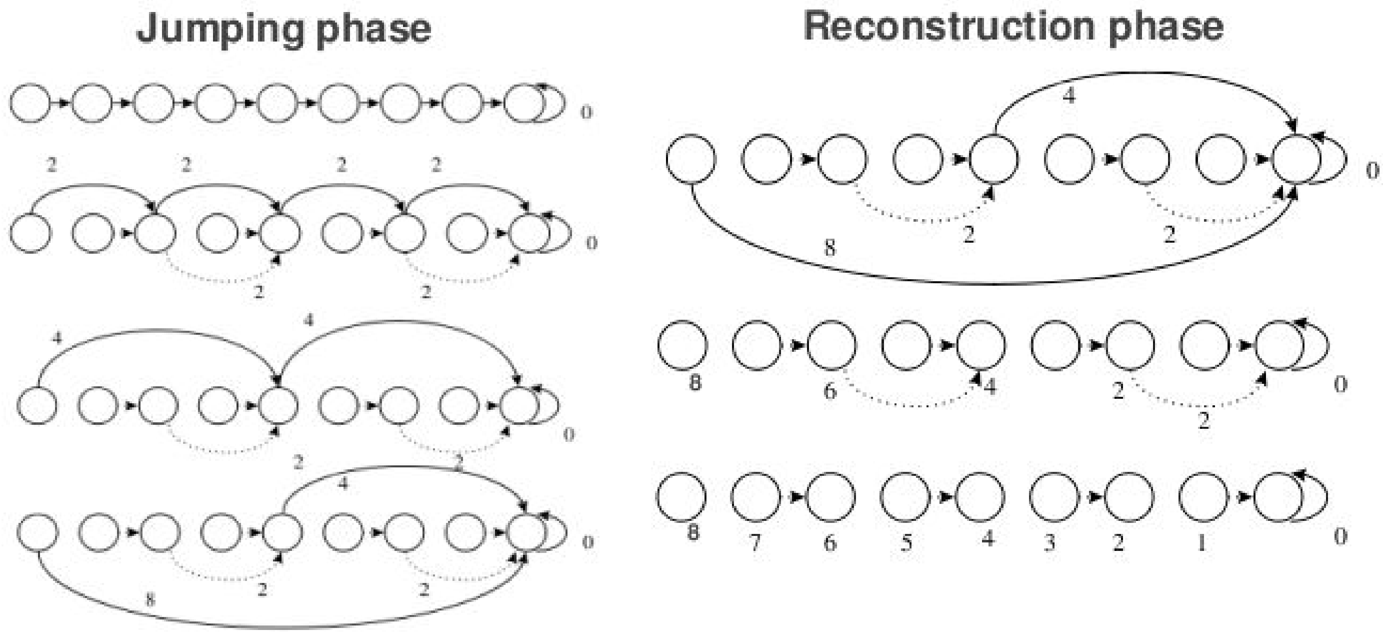
- je obdoba sumy prefixů, ale na seznamech (kde pevný bod je konec)
- Počítá se stejně jako list ranking, ale použije se zadaná operace $\oplus$



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

Optimalizace path doublig a sumy suffixů

- spočívá ve snížení ceny, některé procesory totiž provádějí zbytečnou práci (počítají již spočítané věci nebo



cyklí na konci)

- Řešením je odpojovat procesory a tím snížit cenu (v každém kole odpojena polovina prcesorů)
- Postup:
  - Jumping phase
    - Nejprve každý procesor dostane vzdálenost 1,
    - pak pracuje každý druhý a zvýší ji na 2
    - pak každý čtvrtý a opět zvýší.
  - Recosntruction phase
    - přičítají mezivýsledky.

Problém v paralelismu
: jak se určí "každý druhý" ?

Random mating

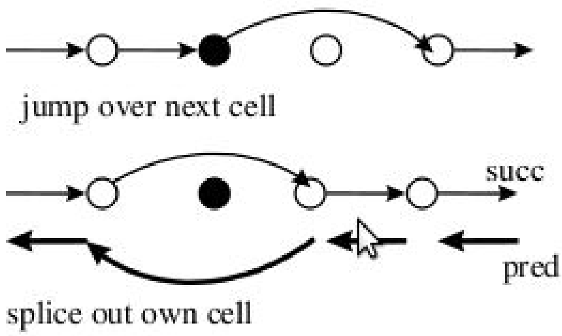
- každý proces si náhodně vybere pohlaví
- každý female následovaný male se přeskočí a procesor se odpojí
- cena je stále neoptimální, ale množství práce je c(n) = O(n) (optimální)

Optimal list ranking

- simulace random mating
- pevný počet stále pracujících procesorů (n/log n procesorů, každý obsluhuje log n prvků)
- každý procesor má zásobník prvků, které má zpracovat
- prvky mají náhodně přiřazeno pohlaví
- v každém kroku se všechny procesory pokusí provést jump-over (přeskočení následníka prvku na vrcholu

zásobníku)
    - (přeskakuje se female jehož následník je male)

- algoritmus může být nevyvážený
- řešení se nahrazením operace jump-over za splice-out (vypletení)



List coloring

List coloring
: je obarvení seznamu tak, aby sousedé neměli stejnou barvu. k-obarvení může použít k různých barev.

$2log(n)$ coloring

- využívá index procesoru k určení barvy.
- Hodnota k je index nejnižšího bitu indexu, ve kterém se sousedé liší, barva je pak $C = 2k + ID[k]$

Ruling set

k-ruling set
: množina nesousedících vrcholů, mezera mezi nimiž je široká maximálně k

$2k$-ruling set z k-coloring

vybere prvek do ruling set tehdy, když jeho barva je nižší než barva předchůdce a následníka (hledáme lokální minima)

Stromy

Stromy
: jsou obvykle prezentovány podobně jako seznamy, ale vazba není na následníka ($i + 1$), nýbrž na syny ($2i$ a $2i + 1$).

Úroveň vrcholu
: počet hran mezi uzlem a kořenem

Eulerova cesta

Eulerova cesta

- je obecný případ průchodu stromem (linearizace)
- průchod stromem ve kterém se každá hrana projde právě jednou (jednou tam a jednou zpátky)

Eulerova kružnice

- strom se převede na orientovaný graf (každá hrana se nahradí za dvě orientované hrany v opačných směrech)
- je reprezentována funkcí `etour(e)`, která hraně přiřadí následující hranu v kružnici.
- Umožňuje projít všechny uzly grafu bez opakování hran na cestě.

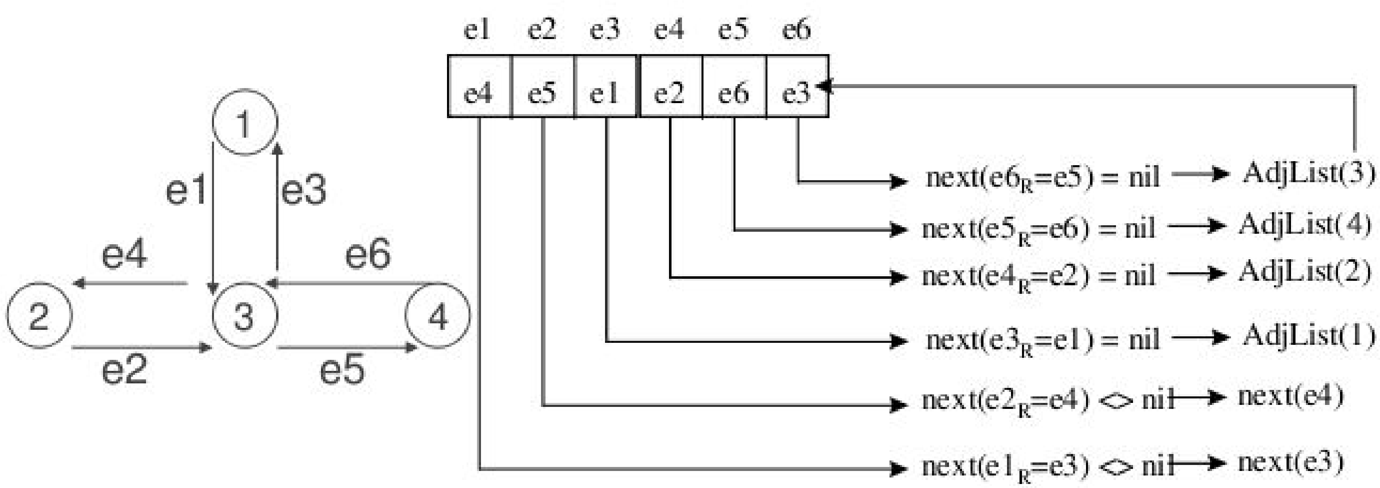
Seznam sousednosti (adjacency list)

- slouží k reprezentaci stromu v podobě grafu pro eulerovu kružnici
- pro každý uzel je přiřazen lineárně vázaný seznam, každý prvek seznamu je dvojice hrana/opačná hrana (e, e_R)

Pozice uzlu
: je vypočtena jako $2n - 2 - Rank(e)$

$Rank(e)$ je výsledek list rankingu - $O(log(n))$. Využívá se pro zjištění rodiče.

Eulerova kružnice




```
for i = 1 to 2n-2 do in parallel     { ei = (u, 
  if next(eR) <> nil  then Etour(e) = next(eR)
    else Etour(e) = AdjList(v) { first item of a
```

```
  endif
endfor
```

Kořen stromu
: vznikne tak, že se v jednom bodě (kořeni) Eulerova kružnice přeruší.

Pozice hran

- vezmeme grav v podobě seznamu souslednosti
- převedeme na Eulerovu kružnici
- provedeme List ranking na kružnici (rank = opačné pořadí hran v kružnici)
- Pořadí získáme paralelním vypočtením 2n-2-rank (n je počet vrcholů)
- t(n) = O(log n) (nejhorší závislost má suma suffixů, ostatní jsou lineární)

Nalezení rodičů

Dopředná hrana
: position(e) < position (e_R)
: tj. hrana na které nejdřív jdeme dopředu a pak až zpátky

Zpětná hrana
: position(e) > position (e_R)
: tj. hrana na které nejdřív jdeme dopředu a pak až zpátky

Pokud (u, v) je dopředná hrana pak u je rodičem v

```
for each edge e = (u, v) do in parallel
  if posn(e) < posn(eR) then
    parent(v) := u;
  endif
  parent(root) := nil;
endfor
```

Preorder

(Preorder – navštiv nejdřív otce, pak oba syny)

- Pořadí preorder vrcholu ve stromě je 1+počet dopředných hran, kterými jsme prošli po cestě z kořene k vrcholu

```
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
```

Počet následníků vrcholu

- Počet dopředných hran v segmentu Eulerovy cesty, počínajícím i končícím ve vrcholu

Úroveň vrcholu

- rozdíl dopředných a zpětných hran na cestě z kořene k vrcholu

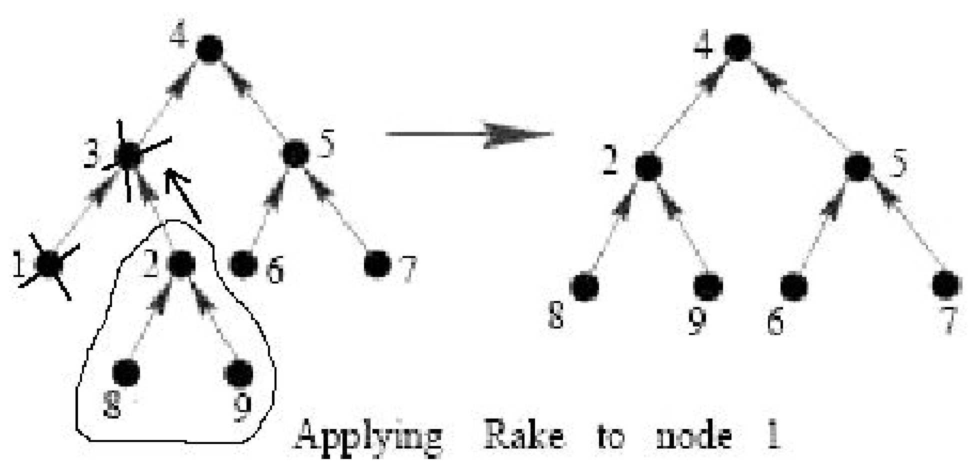
Tree contraction

Tree contraction

- používá se při výpočtu výrazů ve stromě (Eulerova cesta není použitelná)
- Každý list obsahuje operand a nelist operátor
- Tree contraction strom postupně zmenšuje až do jediného uzlu, tedy výsledku.

RAKE operation

- rake na **listový** uzel u
  - odstraníme uzlel u a jeho rodiče
  - druhý potomek rodiče u se připojí na místo, kde byl rodič u



Algoritmus tree contraction

- opakovaně aplikujeme RAKE a tím zmenšujeme strom
- snažíme se aplikovat pro co nejvíce listů paralelně

- nelze aplikovat operaci RAKE na vrcholy jež ve stromu sousedí (mají spol rodiče)
- Jak určit uzly na které lze aplikovat?

- Algoritmus:
  - Označíme listy jejich pořadím zleva doprava (pořadí na Eulerově cestě)
    - 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
  - Uložíme všech n listů do pole A
    - Aodd obsahuje prvky pole A s lichými indexy
    - Aeven obsahuje prvky pole A se sudými indexy
  - for i:=1 to log(n+1) do
    - RAKE na všechny Aodd, které jsou levými potomky
    - RAKE na všechny Aodd, které zbyly
    - A := Aeven

- počet listů se v každém kole změnší na polovinu
- t(n) = O(log n)

Algoritmus Tree search

- Vyhledávání v neseřazené posloupnosti
- Stromová architektura s 2n-1 procesory

- Algoritmus
  - 1. Kořen načte hledanou hodnotu x a předá ji synům ... až se dostane ke všem listům
  - 2. Listy obsahují seznam prvků, ve kterých se vyhledává (každý list jeden). Všechny listy paralelně porovnávají x a xi, výsledek je 0 nebo 1.
  - 3. Hodnoty všech listů se předají kořenu - každý ne list spočte logické or svých synů a výsledek zašle otci.

Kořen dostane 0 - nenalezeno, 1- nalezeno

- Analýza
  - Krok (1) má složitost O(log n), krok (2) má konstantní složitost, krok (3) má O(log n).
  - t(n) = O(log n)
  - p(n) = 2.n-1
  - c(n) = t(n).p(n) = O(n.log n) → což není optimální

obrazek viz PRL H005- str.8

Kategorie: Státnice 2011 | Paralelní a distribuované algoritmy

- Stránka byla naposledy editována 29. 5. 2012 v 14:10.