Voľba šéfa na úplných grafoch so zmyslom pre orientáciu

zmysel pre orientáciu

- HK
- sused identifikovaný vzdialenosťou po HK

algoritmy

- naivný— $O(N^2)$ správ, $O(N)$ čas
- sekvenčné získavanie povolení — $O(N \log N)$ správ, $O(N)$ čas

- 1996 Singh — $O(N)$ správ, $O(\log N)$ čas
- 2000 Dobrev — $O(N)$ správ, $O(\log \log N)$ čas (optimum)

Sekvenčné získavanie povolení – $O(N \log N)$, $O(N)$

správy

$\langle capture, [level, id] \rangle$
$\langle help, [level, id] \rangle$
$\langle accept \rangle$
$\langle victory \rangle$
$\langle defeat \rangle$
$\langle leader, ID \rangle$

stavy

active
killed
dead

- **[level, id]** – v porovnaní vyhráva lexikograficky väčší
- ak jedným z porovnávaných je už porazený vrchol, o výsledku rozhoduje ten, kto ho zajal
- *level* rastie prijatím *accept*
- leadrom sa stáva ten, kto dostal $N - 1$ akceptovaní

Algoritmus sekvenčných porovnaní

Lema 1 *V ľubovoľnom výpočte existuje pre každú úroveň* $l = 0, \ldots, N-1$ *aspoň jeden proces, ktorý bol počas výpočtu na úrovni* l

Lema 2 *Nech* v *je aktívny proces úrovne* l*. Potom existuje* l *procesov, ktoré zajal.*

Lema 3 *V ľubovoľnom výpočte je najviac* $N/(l+1)$ *procesov, ktoré niekedy v priebehu výpočtu boli na úrovni* l*.*

správy $\sum_{l=1}^{N-1} \frac{N}{l+1} = N(H_N - 1) \approx N \log N$

čas po $O(N)$ krokoch niekto na úrovni 1 $\Rightarrow$ zobudený šéf
postup na ďalšiu úroveň v konštantnom čase $\Rightarrow$ $O(N)$

✓

Algoritmus Singh – zmysel pre orientáciu (HK), známa veľkosť siete

$\mathbf{S_i}$

– množina procesov, ktoré je vrchol i schopný zajať a stať sa ich leadrom
– ak ich udržiavame disjunktné, netreba väčšinu ale stačí, aby $\forall i \neq j \; S_i \cap S_j \neq \emptyset$
Nech $k = \frac{N}{2^{\lceil \log \log N \rceil}}$. Leadrom sa stane vrchol, ktorému sa podarí dosiahnuť

$$S_i = \{i[1 \ldots k-1], i[k], i[2k], \ldots, i[N-k]\}$$

$\mathbf{i[j]}$

– vrchol, ktorý je od i na HK vo vzdialenosti j
– $i[0] = i, \; i[1]$ je sused i na HK, ...

$\mathbf{R_i}$ $= \{i[0], i[k], i[2k], \ldots, i[N-k]\}$
– v prvej fáze leader v rámci R_i
– v druhej fáze leader spomedzi nich (*ako vieme, kto je leader kruhu* R_i ?)

Algoritmus Singh – premenné

$state_i$– *passive*, *candidate*(po spontánnom zobudení), *captured*, *elected*

$level_i$– pôvodne 0; označuje aktuálny počet vrcholov zajatých v prvej fáze

$step_i$– pôvodne 0; aktuálny počet krokov kandidáta v druhej fáze

$owner_i$– pre kandiáta je $owner_i = 0$, pre zajatý vrchol je to hrana, ktorá vedie do leadra R_i (vzdialenosť doňho po HK)

$phase_i$– budí sa s fázou 1, do 2 prechádza ako kandidát alebo po prijatí správy fázy 2

Algoritmus Singh –správy

CAPTURE($phase_i, level_i/step_i, i$)– i sa pokúša zajať nový vrchol

INFORM($dist$)– i ňou v druhej fáze oznamuje vrcholu, ktorý ho zajal, vzdialenosť svojho ownera

ACCEPT($phase_i, Level_i$)– akceptuje sa zajatie; v druhej fáze druhý parameter netreba

OWNER(*id*)– na konci prvej fázy leader R_i oznamuje svoju identifikáciu ostatným v R_i

ACK– potvrdzuje prijatie OWNER

ELECTED– na konci druhej fázy leader oznamuje svoju identifikáciu ostatným po spontánnom prebudení i

– $state_i \leftarrow candidate, level_i \leftarrow 0$

– $i \xrightarrow{CAPTURE(1,0,i)} i[k]$

po prijatí CAPTURE(1,l,i) od i po hrane $j(e)$

- ak $state_j \neq candidate \wedge phase_j = 1$ tak $j \xrightarrow{ACCEPT(1,0)} i$
  ak $state_j = passive$ tak $state_j \leftarrow capture$
- ak $state_j = candidate \wedge phase = 1 \wedge (level_j, j) < (l, i)$ tak
  $owner_j \leftarrow e$ $j \xrightarrow{ACCEPT(1,level_j)} i$ inak sa správa ignoruje
- ak $phase_j = 2$ tak sa táto správa ignoruje

po prijatí ACCEPT(1,l) od j

- ak je ešte stále kandidát $level_i \leftarrow level_i + l + 1$

- ak $level_i < N/k$ pokračuje $i \xrightarrow{CAPTURE(1,level_i)} i[(level_i + 1)k]$
- inak $i \xrightarrow{OWNER(i)} j, \forall j \in R_i$

po prijatí OWNER(i) od i po hrane $j(e)$

- ak už je zajatý vrcholom v druhej fáze, ignoruje
- inak si nastaví $owner_j \leftarrow i$ a pošle $j \xrightarrow{ACK(j)} i$po hrane $j[owner_j]$

po prijatí ACK(j) od $\forall j$

ak je i ešte kandidát, prechádza do druhej fázy

Ako zabezpečíme porovnanie (jediných) kandidátov z jednotlivých R_i?

- i posiela správu 2^{l-1} "**kandidátom**"z kruhov $R_{i[k/2^l]}, R_{i[3k/2^l]}, \ldots, R_{i[(2^l-1)k/2^l]}$
- ak ich všetkých zajme, zvyšuje si krok
- ak $step_i > \log k$, i sa stáva leadrom

i sa pokúša zajať $i[1 \ldots k-1]$

po prijatí CAPTURE(2,l,i) od i vo vrchole $j = i[x]$

- ak $state_j = candidate, phase_j = 2, (l,i) > (step_j, j)$ tak
  - $j \xrightarrow{ACCEPT(2)} i;\ state_j \leftarrow captured$
- ak $state_j = candidate, phase_j = 2, (l,i) < (step_j, j)$ tak j správu ignoruje
- ak $(state_j = candidate \wedge phase_j = 1)\ OR\ (state_j \neq candidate \wedge owner_j = 0)$ tak $j \xrightarrow{ACCEPT} i$
- ak $state_j \neq candidate \wedge owner_j \neq 0$ tak $j \xrightarrow{INFORM(owner_j)} i$

po prijatí INFORM(y) cez $i[x]$

$$i \xrightarrow{CAPTURE(2,step_i,i)} i[x+y]$$

$\mathbf{cap(l, i)} = \{\mathbf{i[k], i[2k], \ldots, i[l \cdot k]}\}$

Lema 4 *Ak* $level_i > 0 \wedge phase_i = 1$ *tak*

1. *v* $cap(level_i, i)$ *neexistuje iný kandidát*
2. $\forall j \in cap(level_i, i):\ cap(level_j, j) \subseteq cap(level_i, i) \wedge level_j < level_i$

Lema 5 *V rámci* $O(N/k)$ *krokov od zobudenia sa prvého vrchola niektorý z vrcholov postúpi do druhej fázy.*

Lema 6

- *Nanajvýš* $k/2^{l-1}$ *kandidátov môže v druhej fáze dosiahnuť* $step = l$
- *V rámci* $O(\log k)$ *krokov odvtedy, ako prvý vrchol dosiahne druhú fázu, sa niektorý vrchol prehlási za leadra*

Lema 7

- *každý vrchol je v prvej fáze priamo zajatý nanajvýš raz*
- *počas prvej fázy sa v každom podcykle pošle* $O(N/k)$ *správ*
- *v rámci druhej fázy sa pošle* $O(k \log k)$ *správ*

Algoritmus Dobrev – asynchrónne orientované úplné siete

1. spontánne zobudený vrchol–kandidát začína v kole 1
2. kandidát v kole k sa pokúša obsadiť súvislú susednú oblasť
3. ak sa kandidátovi podarí obsadiť oblasť veľkosti $> N/2$, stáva sa leadrom

$\mathbf{S_k}$ kandidát sa v $k-$tom kole pokúša obsadiť oblasť veľkosti S_k^2, pričom použije iba $O(S_k)$ správ.

- $S_1 = O(1)$, napr. $S_1 = 4$
- $S_{k+1} = S_k^2 / 2^k$

$$S_k = \frac{S_1^{2^{k-1}}}{2^{\sum_{i=1}^{k} i 2^{k-i}}} \in \Theta\left(\frac{4^{2^k}}{2^{2^k}}\right) = \Theta(2^{2^k})$$

- do kola $k+1$ postúpi nanajvýš N/S_k^2 kandidátov
- cena $k+1$-ého kola je $O(S_{k+1} \cdot N/S_k^2) = O(N/2^k)$
- počet kôl nanajvýš $O(\log\log N)$, cena prvého kola $O(S_1 N) = O(N)$
- sumáciou cez všetky kolá dostaneme $O(N)$

zajatie oblasti veľkosti S_k^2

- najprv súvislý blok veľkosti S_k
- potom S_k procesorov naľavo a napravo vo vzdialenosti S_k

$\rightsquigarrow$ **pozor na časovanie**

správy ("*type*", id, r), kde

– $type \in \{target, bullet, die!, OK, alive, dead\}$
– id je číslo procesora
– r je číslo kola

premenné procesor si pamätá

– poslednú správu typu *target, alive, dead*
– najsilnejšiu správu, ktorú videl -v M
(porovnávame najprv kolo, potom $bullet > target$, potom id)

spontánne zobudený procesor v začína s kolom $r = 0$

while $S_r^2 \leq N/2$ **do**
  $r \leftarrow r + 1;$
  send $(target, id, r)$ po linkách $\pm 1, \pm 2, \ldots, \pm S_r/2;$
  čakaj na $die!$ alebo $OK;$
  if die! **then**
    send $(dead, id, r)$ po linkách $\pm 1, \pm 2, \ldots, \pm S_r/2;$
    terminuj
  else $\triangleright$ OK
    send $(alive, id, r)$ po linkách $\pm 1, \pm 2, \ldots, \pm S_r/2;$
    send $(bullet, id, r)$ po linkách $\pm S_r, \pm 2S_r, \ldots, \pm S_r^2/2;$
    čakaj na $die!$ alebo $OK;$
    if die! **then** terminuj
    end if
  end if
end while
send $(Leader : id)$ všetkým procesom

po prijatí správy $m = (target, id, r)$ v procesore v po linke l

if m je slabšia ako M **then** send $(die!)$ po l
else send (OK) po l
end if

po prijatí správy $m = (bullet, id, r)$ v procesore v po linke l

if m je slabšia ako M **then** send $(die!)$ po l
else
  if v dostalo $(target, id', r)$ s $id' > id$ **then**
    čakaj $(dead, id', r)$ alebo $(alive, id', r)$;
    if (alive, id',r) **then** send $(die!)$ po l
    end if
  end if
  send (OK) po l
end if

Lema 8 *Nech u, v sú dvaja kandidáti, ktorí prežili kolo r. Potom ich vzdialenosť (po HK) je viac ako S_r^2*

Veta 1 *Algoritmus Algoritmus Target & Bullets zvolí šéfa v asynchrónnom orientovanom úplnom grafe, pričom*

čas	$O(\log \log N)$
počet správ	$O(N)$

<u>Čas</u>

- $S_r \in O(2^{2^r})$, preto stačí $O(\log \log N)$ kôl
- Nech t_r je čas, keď prvý kandidát dosiahol kolo r. Ukážeme, že $t_{r+1} \leq t_r + 6$

  u – prvý postúpil do r
  v – prvý vystrelil *bullet* v čase t'_r; $t'_r \leq t_r + 2$
  w – prvý ukončil r v čase t_{r+1};
  $t_{r+1} \leq t'_r + 2 + c$, kde c je čakanie na $die!, OK$
  $c \leq 2 \quad \Rightarrow \quad t_{r+1} \leq t'_r + 4 \leq t_r + 6$

$\square$

Detekcia ukončenia - výpočet terminuje, ak DS dosiahne terminálnu konfiguráciu.

explicitné proces samotný rozpozná, že je v terminálnej konfigurácii; tzv. *proces termination*

implicitné proces by mohol pokračovať prijatím správy, ale v systéme sa správy nenachádzajú. Proces nevie o terminovaní; tzv. *message termination*

Základný algoritmus:

var $state_p : \{active, passive\}$

S_p: $\{state_p = active\}$
begin send $\langle$**mes**$\rangle$ **end**

R_p: {do p dorazila správa $\langle$**mes**$\rangle$ }
begin receive $\langle mes \rangle$; $state_p := active$ **end**

I_p: $\{state_p = active\}$
begin $state_p := passive$ **end**

Fakt 1

$$\begin{aligned}\boldsymbol{term} \Leftrightarrow\ & (\forall p \in P:\ \ state_p = passive) \wedge \\ & (\forall pq \in E : M_{pq}\ neobsahuje\ spr\acute{a}vu\ \langle \boldsymbol{mes} \rangle)\end{aligned}$$

Kontrolný algoritmus pozostáva z *algoritmu detekcie ukončenia (**ADU**)* a *algoritmu oznámenia ukončenia(**AOU**)*.

Algoritmus **d**etekcie **u**končenia musí splĺňať nasledujúce podmienky

- **nezasahovanie** do výpočtu základného algoritmu
- **životnosť** – keď začne platiť **term**, musí sa v konečnom počte krokov zavolať AOU
- **bezpečnosť** – keď sa volá AOU, musí platiť **term**

Dolné odhady

N	počet vrcholov siete
$\lvert E \rvert$	počet hrán/kanálov
M	počet správ poslaných základným algoritmom
W	počet správ poslaných najlepším vlnovým algoritmom

Veta 2 *Ku každému algoritmu detekcie ukončenia existuje výpočet základného algoritmu, v ktorom sa pošle M správ a pre ktorý algoritmus detekcie pošle aspoň M správ.*

Veta 3 *Detekcia ukončenia decentralizovaného výpočtu základného algoritmu vyžaduje v najhoršom prípade výmenu aspoň W kontrolných správ.*

C1 kanály sú FIFO
C2 ADU môže začať v ľubovoľnej konfigurácii základného algoritmu

var $SendStop_p$:boolean init false;
$RecStop_p$:integer init 0;

Procedure Announce:

begin
if not $SentStop_p$
then begin
$SentStop_p$:=true;
forall $q \in Out_p$ do send $\langle$ stop $\rangle$ do q
end
end

{ po prijatí $\langle stop \rangle$ }
receive $\langle stop \rangle$, $RecStop_p \leftarrow RecStop_p + 1$
Announce
if $RecStop_p$ =počet prichádzajúcich hrá
then
halt
end if

difúzny, p_0 je **iniciátor**, ADU udržiava **výpočtový strom** $T = (V_T, E_T)$

- T je alebo prázdny, alebo orientovaný s koreňom p_0
- V_T obsahuje všetky aktívne procesy a prenášané správy

Iniciátor volá Announce, keď je T prázdny

Dijkstra-Scholten

(Pre proces p) sc_p označuje počet jeho detí v strome T a počet odchádzajúcich správ.

- Keď p pošle správu, $sc_p := sc_p + 1$
- Nech túto správu príjme q
  - ak q ešte nebolo v T, q sa stane vrcholom v T, jeho otcom bude p, $sc_q := 0$
  - ak už q je v T, pošle vrcholu p správu, že nie je jeho novým synom. Po prijatí tejto správy si p obnoví info: $sc_p := sc_p - 1$
- Keď sa neiniciátor p stane pasívnym a $sc_p = 0$, informuje otca o tom, že už nie je jeho synom
- Keď sa pasívnym stane iniciátor a $sc_{p_0} = 0$, zavolá *Announce*

KOREKTNOSŤ: Pre každú konfiguráciu γ definujme

$$V_T = \{p : father_p \neq undef\} \cup \{\langle mes, p\rangle \text{ in transit }\} \cup \{\langle sig, p\rangle \text{ in transit }\}$$

$$\begin{array}{rcl} E_T & = & \{(p, father_p){:}father_p \neq undef \wedge father_p \neq p\} \\ & \cup & \{(\langle mes, p\rangle, p){:}\langle mes, p\rangle \text{ in transit }\} \\ & \cup & \{(\langle sig, p\rangle, p){:}\langle sig, p\rangle \text{ in transit }\} \end{array}$$

Fakt 2 *Invariantom algoritmu je*

$$\begin{aligned} P \equiv \quad & state_p = active \Rightarrow p \in V_T & (1) \\ \wedge \ & (u, v) \in E_T \Rightarrow u \in V_T \wedge v \in V_T \cap P & (2) \\ \wedge \ & sc_p = \#\{v : (v, p) \in E_T\} & (3) \\ \wedge \ & V_T \neq \emptyset \Rightarrow T \text{ je strom s koreňom } p_0 & (4) \\ \wedge \ & (state_p = passive \wedge sc_p = 0) \Rightarrow p \notin V_T & (5) \end{aligned}$$

Fakt 3 $\#$ *kontrolných správ* $\leq \#$ *základných správ.*

Veta 4 *Algoritmus Dijkstra-Sholten je korektným algoritmom detekcie ukončenia, ktorý používa M kontrolných správ.*

Shavit-Francez Základný algoritmus môže byť *decentralizovaný*, kanály sú *obojsmerné*

Udržiavame les F orientovaných stromov, koreňom sú iniciátori. Každý proces je nanajvýš v jednom strome.

- Keď p pošle správu, $sc_p := sc_p + 1$
- Nech túto správu príjme q
  - ak q ešte nebolo v niektorom strome v F, q sa stane vrcholom v F, jeho otcom bude p, $sc_q := 0$
  - ak už q je vrcholom v F, pošle vrcholu p správu, že nie je jeho novým synom. Po prijatí tejto správy si p obnoví info: $sc_p := sc_p - 1$
- Keď sa neiniciátor p stane pasívnym a $sc_p = 0$, informuje otca o tom, že už nie je jeho synom

Na pozadí beží vlnový algoritmus. Iniciátor sa môže zapojiť do jeho výpočtu len vtedy, keď jeho strom je prázdny. Rozhodnutie zavolá *Announce*

$\mathbf{S_p}$ $\triangleright\ state_p = active$

send $\langle mes, p\rangle; sc_p \leftarrow sc_p + 1$

$\mathbf{R_p}$ $\triangleright$ správa $\langle mes, q\rangle$ dorazila do p

receive $\langle mes, q\rangle; state_p \leftarrow active$
if $father_p = undef$ **then** $father_p \leftarrow q$
else send $\langle sig, q\rangle$ do q
end if

$\mathbf{I_p}$ $\triangleright\ state_p = active$

$state_p \leftarrow passive$
if $sc_p = 0$ **then** $\triangleright$ odstránenie p z F
  if $father_p = p$ **then** $empty_p \leftarrow true$
  else send $\langle sig, father_p\rangle$ do $father_p$
  end if
  $father_p \leftarrow undef$
end if

$\mathbf{A_p}$ $\triangleright$ do p prišla správa $\langle sig, p\rangle$

receive $\langle sig, p\rangle;\ sc_p \leftarrow sc_p - 1$
if $(sc_p = 0) \& (state_p = passive)$ **then**
  if $father_p = p$ **then** $empty_p \leftarrow true$
  else send $\langle sig, father_p\rangle$ do $father_p$
  end if
  $father_p \leftarrow undef$
end if

Veta 5 *Algoritmus Shavit-Francez je korektným algoritmom detekcie ukončenia, ktorý používa $M + W$ kontrolných správ.*

$$V_F = \{p : father_p \neq undef\} \cup \{\langle mes, p\rangle \text{ in transit }\} \cup \{\langle sig, p\rangle \text{ in transit }\}$$

$$\begin{aligned} E_F \;=\;& \{(p, father_p){:}father_p \neq undef \wedge father_p \neq p\} \\ \cup\;& \{(\langle mes, p\rangle, p){:}\langle mes, p\rangle \text{ in transit }\} \\ \cup\;& \{(\langle sig, p\rangle, p){:}\langle sig, p\rangle \text{ in transit }\} \end{aligned}$$

Invariant

$$\begin{aligned} Q \Leftrightarrow \quad & state_p = active \Rightarrow p \in V_F && (1) \\ \wedge \quad & (u, v) \in E_F \Rightarrow u \in V_F \wedge v \in V_F \cap P && (2) \\ \wedge \quad & sc_p = \#\{v; (v, p) \in E_F\} && (3) \\ \wedge \quad & V_F \neq \emptyset \Rightarrow F \ je \ les && (4) \\ \wedge \quad & (state_p = passive \wedge sc_p = 0) \Rightarrow p \notin V_F && (5) \\ \wedge \quad & empty_p \Leftrightarrow T_p = \emptyset && (6) \end{aligned}$$

Dijkstra-Feijen-Van Gasteren

$$\mathbf{term} \Leftrightarrow \forall p : state_p = passive$$

Proces p_0 je iniciátorom traverzovacieho algoritmu, aby zistil, či sú všetky procesy pasívne.

Komplikácia 1: Nespracované správy Riešenie: *Synchrónna komunikácia*

$\mathbf{C_{pq}}$: $\triangleright\ state_p = active$

$state_q \leftarrow active$ $\triangleright$ p posiela základnú správu pre q

$\mathbf{I_p}$:
$state_p \leftarrow passive$

Komplikácia 2: Traverzovanie iba cez pasívne procesy nezaručuje terminovanie

Riešenie: ***Farbenie procesov*** na bielo a čierno. Na začiatku sú biele, po poslaní základnej správy (správy v základnom algoritme) sa prefarbia na čierno.

- Keď je p_0 pasívny, posiela biely token
- Token je posúvaný len pasívnymi procesmi
- Ak token posúva čierny proces, stane sa token čiernym a proces bielym
- Ak sa token vráti do p_0, p_0 počká, kým bude pasívny
  - ak aj token aj proces sú biele, p_0 zavolá *Announce*
  - inak p_0 opäť pošle biely token

  Prefarbovanie tokena namiesto poslania je neprípustné

Algoritmus Dijkstra-Feijen-Van Gasteren

$\mathbf{C_{pq}}$ $\quad\triangleright$ $state_p = active$
$color_p \leftarrow black;\ state_p \leftarrow active$

$\mathbf{I_p}$ $\quad\triangleright$ $state_p = active$ $state_p \leftarrow passive$
detekciu začína send $\langle tok, white\rangle$ do p_{N-1}

$\mathbf{T_p}$ $\quad\triangleright$ spracovanie $\langle tok, c\rangle$

```
if p = p_0 then
    if (c = white ∧ color_p = white) then
        Announce
    else send ⟨tok, white⟩ do p_{N-1}
    end if
else if color_p = white then
    send ⟨tok, c⟩ do Next_p
else send ⟨tok, black⟩ do Next_p
end if
color_p ← white
```

$P_0 \equiv \forall i(N > i > t) : state_{p_i} = passive$
$P_1 \equiv \exists j(t \geq j \geq 0) : color_{p_j} = black$
$P_2 \equiv token\ je\ black$

Fakt 4 *Počet správ pre DFG je $O(T + |E|)$, kde T je zložitosť základného algoritmu.*

Korektnosť: Cez *invariant* $P = P_0 \vee P_1 \vee P_2$

Safra - počítanie správ – *aj jednosmerné kanály a asynchrónne posielanie správ*

Nech B je počet prenášaných správ. Potom **term** $\equiv (\forall p : state_p = passive) \wedge B = 0$

Každý proces si udržiava počítadlo, ktoré je na začiatku 0.

- Keď je p_0 pasívny, pošle biely token
- Token posúvajú iba pasívne procesy. Keď proces posúva token, pripočíta hodnotu svojho počítadla k počítadlu tokena (*Rule 1*)
- Prijatím správy sa proces stáva čiernym (*Rule 2*)
- Poslaním tokena sa proces stáva bielym (*Rule 5*)
- Ak posúva token čierny proces, token sa stane čiernym (*Rule 3*)
- Keď sa token vráti do p_0, p_0 počká, kým bude pasívny.
  - Ak sú aj token aj p_0 biele a suma všetkých počítadiel je 0, p_0 zavolá *Announce*
  - inak pošle p_0 opäť biely token (*Rule 4*)

Posielaný token sa stane čiernym len posielaním, nesmie sa len prefarbiť.

$\mathbf{S_p}$: send $\langle mes \rangle$; $mc_p \leftarrow mc_p + 1$

$\mathbf{R_p}$:
receive $\langle mes \rangle$; $state_p \leftarrow active$
$mc_p \leftarrow mc_p - 1$; $color_p \leftarrow black$

$\mathbf{I_p}$: $state_p \leftarrow passive$

$$
\begin{aligned}
P_m &\equiv B = (\textstyle\sum_{p \in P} mc_p) \\
P_0 &\equiv (\forall i (N > i > t) : state_{p_i} = passive) \\
&\wedge (q = \textstyle\sum_{N>i>t} mc_{p_i}) \\
P_1 &\equiv (\textstyle\sum_{N>i>t} mc_{p_i} + q) > 0 \\
P_2 &\equiv \exists j (t \geq j \geq 0) : color_{p_j} = black \\
P_3 &\equiv \text{token je black}
\end{aligned}
$$

invariant $P = P_m \wedge (P_0 \vee P_1 \vee P_2 \vee P_3)$

$\mathbf{T_p}$:
if $p = p_0$ **then**
  if $(c = white \wedge color_p = white \wedge mc_p + q = 0)$ **then** Announce
  else send $\langle tok, white, 0 \rangle$ do p_{N-1}
  end if
else
  if $color_p = white$ **then** send $\langle tok, c, q + mc_p \rangle$ do $Next_p$
  else send $\langle tok, black, q + mc_p \rangle$ do $Next_p$
  end if
end if
$color_p \leftarrow white$

Veta 6 *Algoritmus* ***Safra*** *korektne rieši problém detekcie ukončenia pre výpočty s asynchrónnym posielaním správ.*

Mattern navrhol podobný algoritmus ako Safra, ale použil **vektorové počítadlo**

výhody

- rýchlejšia detekcia ukončenia; jedno kolo po terminovaní
- kým nedôjde k terminovaniu, častejšie sa stane, že je token zastavený; toto vlastne znižuje počet posielaných správ. Safra nechal každý pasívny proces posúvať token, pri vektorovom počítadle pasívny proces neposúva, ak je z počítadla zrejmé, že jeho správa ešte nedorazila na miesto určenia.

nevýhody

- veľká pamäť tokena {ak ich nie je veľa, nie je to zlé}
- potrebujeme identifikáciu procesov {môžme vyrábať priebežne; namiesto poľa zoznam dvojíc}

Optimalizovaný Dijkstra & Safra

$\boxed{\textbf{I.}}$ poslanie správy prefarbuje na čierno proces aj keď netreba — **enumeračné bity**

- iniciovaný s hodnotou 0
- prechod tokena mení hodnotu $\mathbf{0} \leftrightarrow \mathbf{1}$
- do správy pridáme id a enumeračný bit
- P_j prijíma správu $m, id_m, enum_m$ — P_j sa prefarbí len ak

$$enum_j \neq enum_m \quad \& \quad id_j > id_m$$

$\boxed{\textbf{II.}}$ namiesto jediného iniciátora sú iniciátormi **všetci**

- vektor súčtov
- vektor enumeračných bitov
- čierny proces (P_2) prefarbí na čierno všetky sumy, spustí novú detekciu — $[b, 0, b, \ldots, b]$

dobré idey, ale nestačia $\rightsquigarrow$ spojenie

(Optimálna) Detekcia ukončenia na kruhu - v procese P_j

send msg

$c.j \leftarrow c.j + 1$
send$\langle msg, enum.j, j\rangle$

receive msg

$c.j \leftarrow c.j - 1$
if $enum.j \neq enum.m \;\wedge\; id.m < j$ **then**
  for all $k : j \leq k \leq N \;\wedge\; 1 \leq k < id_m$ **: do**
    $color.j.k \leftarrow black$
  end for
end if
receive m

posun tokena

$//idle.j \;\wedge\; \neg(q.j + c.j = 0 \;\wedge\; token\_color.j = white \;\wedge\; color.j.j = white)$

for all k **do**
  if $color.j.k = black$ **then** $token\_color.k \leftarrow black$
  end if
end for
for all $k : token\_color.k \neq black$ **do**
  $q.k \leftarrow q.k + c.j$
end for
$q.j \leftarrow 0$; $token\_color.j.k \leftarrow white$
for all k **do**$color.j.k \leftarrow white$
end for
$enum.j \leftarrow \neg(enum.j)$
send token do $(j \bmod N) + 1$

X – detekcia
W – witness
I – invariant

$\boxed{W \text{ detekuje } X}$ ak

- $W \Rightarrow X$
- X vedie (v budúcnosti) k W

Predikát detekcie X

$X = (\forall j : idle.j) \ \wedge \ (\#sent - \#received = 0)$

Predikát witness W

$(\exists j : \text{token v } j) \ \wedge \ (idle.j) \ \wedge \ (color.j = white) \ \wedge \ (c.j + q.j = 0) \wedge$
$(token\_color.j = white)$

Invariant $I =$

$$(\sum_j : c.j = \#sent - \#received) \wedge (\forall \text{ inic } (Q.inic) \vee (R.inic) \vee (S.inic) \vee (T.inic))$$

Q.inic=$(\forall j \text{ v regione iniciátora: } idle.j) \wedge (q.inic = \sum_{\text{j v regione iniciátora}} c.j)$
R.inic=$q.inic + (\sum_{\text{j v nenavštívenom regione iniciátora}} c.j) > 0$
S.inic=$(\exists \text{j v nenavštívenom regione iniciátora} : color.j.inic = black)$
T.inic=$token\_color.inic$ je $black$

Využitie vĺn

Pri konštrukcii ADU využijeme ľubovoľný vlnový algoritmus.
Pre každú vlnu bude **visit procesu** prvá udalosť, ktorou v nej proces poslal správu, resp. ktorou rozhodol. Ak treba, môže s touto udalosťou počkať, kým nenastanú nejaké špecifické podmienky.

1. Vlna prechádza len cez pasívne procesy
2. Poslaním správy sa proces zafarbí na čierno
3. Navštívený proces odovzdá vlne svoju farbu
4. Rozhodnutie v čiernej vlne vyvolá novú vlnu
5. Po navštívení sa proces stáva bielym

Toto však predpokladá, že máme jedinú vlnu. V decentralizovanom prípade treba aby aj vlnový algoritmus bol decentralizovaný.

Využitie kreditu-Mattern

Idealizovaný prípad

- centralizovaný výpočet, iniciátor je v strede hviezdy (priame spojenie na iniciátora);
- proces môže v nulovom čase vykonať ľubovoľný konečný počet udalostí a čas medzi poslaním a prijatím správy je nanajvýš jednotkový

Aj správa aj proces dostanú **kredit**$\in \langle 0, 1 \rangle$

S1 súčet kreditov je vždy 1

S2 základná správa má kladný kredit

S3 aktívny proces má kladný kredit, pasívny proces má kredit=0

Každý proces(ak mu to ostatné pravidlá umožnia) oznamuje svoj kredit iniciátorovi. Ten na základe toho vie rozhodnúť, či došlo k terminovaniu.

Algoritmus Credit-Recovery

state$_\mathbf{p}$	$active_p, passive$	if $p = p_0$ then active, else passive
credit$_\mathbf{p}$		if $p = p_0$ then 1 else 0
ret		len p_0 0

S$_\mathbf{p}$: send $\langle mes, credit_p/2\rangle$; $credit_p \leftarrow credit_p/2$ $\quad \triangleright$ $state_p = active$

R$_\mathbf{p}$: receive $\langle mes, c\rangle$; $state_p \leftarrow active$; $credit_p \leftarrow credit_p + c$ $\quad \triangleright$ po prijatí $\langle mes, c\rangle$ od q

I$_\mathbf{p}$: $state_p \leftarrow passive$; send $\langle ret, credit_p\rangle$ do p_0; $credit_p \leftarrow 0$ $\quad \triangleright$ $state_p = active$

A$_\mathbf{p0}$:
receive $\langle ret, c\rangle$; $ret \leftarrow ret + c$; $\quad \triangleright$ po prijatí $\langle ack\rangle$v p
if $ret = 1$ **then** Announce
end if

R1 Ak $ret = 1$, iniciátor zavolá *Announce*

R2 Keď sa proces stane pasívnym, pošle svoj kredit iniciátorovi

R3 Keď proces pošle správu procesu p, rozdelí sa jeho kredit medzi správu a p

R4 Pri aktivácii dostane proces kredit správy, ktorá ho aktivovala

R5-alternatívne
- a, Ak aktívny proces príjme správu, pošle kredit tejto správy iniciátorovi;
- b, Ak aktívny proces príjme správu, pripočíta kredit tejto správy ku kreditu procesa.

Veta 7 *Algoritmus credit-Recovery korektne rieši problém detekcie ukončenia.*

Pre potreby tohto algoritmu predpokladáme, že procesy majú hodiny(napríklad Lamportove logické hodiny). Snažíme sa zistiť, či v nejakom časovom okamihu t boli všetky procesy v kľude.

Realizujeme to vlnou, ktorá žiada potvrdiť, že bol proces v čase t a neskôr v kľude. Ten, ktorý nebol, neodpovie, čím vlnu zastaví.

Vlna nezasahuje do algoritmu na určenie ukončenia, viaceré vlny algoritmu neprekážajú.

Algoritmus je decentralizovaný. Proces si v premennej qt_p uchová okamih, v ktorom ostal v kľude. Súčasne spustí vlnu, aby overil, ako sú na tom ostatné procesy.

$$\langle\ tok,\ \Theta,\ qt,\ q\ \rangle$$

Algoritmus Rana

$\mathbf{S_p}$: $\Theta_p \leftarrow \Theta_p + 1$; send $\langle mes, \Theta_p \rangle$; $unack_p \leftarrow unack_p + 1$

$\mathbf{R_p}$: $\triangleright$ po prijatí $\langle mes, \Theta \rangle$

receive $\langle mes, \Theta \rangle$; $\Theta_p \leftarrow \max\{\Theta_p, \Theta\} + 1$; send $\langle ack, \Theta_p \rangle$ do q; $state_p \leftarrow active$

$\mathbf{I_p}$:

$\Theta_p \leftarrow \Theta_p + 1; state_p \leftarrow passive$

if $unack_p = 0$ **then** $qt_p \leftarrow \Theta_p$; send $\langle tok, \Theta_p, qt_p, p \rangle$ do $Next_p$

end if

$\mathbf{A_p}$: $\triangleright$ po prijatí $\langle ack, \Theta \rangle$

receive $\langle ack, \Theta \rangle$; $\theta_p \leftarrow \max\{\Theta_p, \theta\} + 1; unack_p \leftarrow unack_p - 1$

if $unack_p = 0$ a $state_p = passive$ **then**

  $qt_p \leftarrow \Theta_p$; send $\langle tok, \Theta_p, qt_p, p \rangle$ do $Next_p$

end if

$\mathbf{T_p}$: $\triangleright$ po prijatí $\langle tok, \Theta, qt, q \rangle$

receive $\langle tok, \Theta, qt, q \rangle$; $\Theta_p \leftarrow \max\{\Theta_p, \Theta\} + 1$

if $quiet(p)$ **then**

  if $p = q$ **then** Announce

  else if $qt \geq qt_p$ **then** send $\langle tok, \Theta, qt, q \rangle$ do $Next_p$

  end if

end if

Veta 8 *Algoritmus Rana korektne rieši problém detekcie ukončenia.*

$quiet(p) \Rightarrow$ $state_p = passive \wedge$ v systéme neexistuje nespracovaná správa poslaná procesom p

Preto ($\forall p \; quiet(p) \Rightarrow \mathbf{term}$),

pričom $quiet(p) \equiv (state_p = passive \wedge unack_p = 0)$