Algoritmy výberu - Voľba šéfa - election/leader election

Problém distribuovaného výberu (voľba šéfa) vyžaduje prejsť z konfigurácie, kde sú všetky procesory v stave začiatok do konfigurácie, kde práve jeden procesor je v stave šéf a ostatné v stave nešéf.

- každý proces pracuje podľa toho istého lokálneho algoritmu
- algoritmus je decentralizovaný
- v každej realizácii dosiahne algoritmus terminálnu konfiguráciu, v ktorej je jediný proces v stave `šéf-leader` a všetky ostatné procesy sú v stave `nešéf-lost`

Predpoklady:

- asynchrónnosť
- proces je jednoznačne určený svojim menom(z úplne usporiadanej množiny P); w označuje počet bitov, ktoré treba na zápis mien
- niektoré algoritmy sú z triedy porovnávacích
- veľkosť prenášaných správ je $O(w)$

Výber a vlny - Výber na strome

Keďže pri stromovom algoritme potrebujeme, aby ho spustili všetky listy, pridáme fázu - **wake-up**, ktorou sa vrcholy prebudia. Iniciátor rozpošle správu $\langle wakeup \rangle$. V premennej ws si proces pamätá, či už poslal $\langle wakeup \rangle$, v premennej wr zas počet prijatých $\langle wakeup \rangle$.

Algoritmus ET

1. inicializácia
   $ws_p \leftarrow false; wr_p \leftarrow 0; rec_p[q] \leftarrow false; v_p \leftarrow p; state_p \leftarrow sleep$
2. wakeup fáza
3. vlna na strome

Tvrdenie 1 *Algoritmus ET rieši voľbu šéfa na stromoch.*
počet správ - $O(N)$
čas - $O(D)$

Počet správ môžme zredukovať na $3N - 4 + k$, kde k je počet iniciátorov - "nelistov".

Výber na orientovanom kruhu

LeLann,1977	$\longrightarrow$	počet správ $O(n^2)$
Chang a Roberts, 1979	$\longrightarrow$	priemerný počet správ $O(nlogn)$
Hirschberg-Sinclar,1980	$\longleftrightarrow$	horný odhad $O(nlogn)$; vyžaduje, aby kanály boli obojsmerné
Petersen,82 / Dolev,Klawe a Rodeh,82	$\longrightarrow$	horný odhad $O(nlogn)$

LeLann - jednosmerný kruh

Každý procesor eviduje čísla identifikátorov, ktoré videl. Na záver sa leadrom stáva mini-
mum. Vďaka tomu, že identifikačné čísla procesorov sú jednoznačné, procesor identifikuje
koniec podľa toho, že sa mu vrátil jeho token.

Tvrdenie 2 *Algoritmus LeLann rieši problém voľby*
počet správ - $O(N^2)$
čas - $O(N)$

Algoritmus LeLann

z pohľadu p

- inicializácia - $List_p \leftarrow \{p\}$
- **iniciátor**
  $state_p \leftarrow cand; send\langle tok, p\rangle$ do $Next_p; receive\langle tok, q\rangle$
  while q $\neq$ p do begin
  $List_p \leftarrow List_p \cup \{q\}$
  send $\langle tok, q\rangle$ do $Next_p; receive\langle tok, q\rangle$
  end;
  if $p = \min(List_p)$ then $state_p \leftarrow leader$ else $state_p \leftarrow lost$
- **neiniciátor**
  repeat
  $receive\langle tok, q\rangle; send\langle tok, q\rangle$ do $Next_p;$
  if $state_p = sleep$ then $state_p \leftarrow lost$
  until $false$

Algoritmus Chang-Roberts — jednosmerný decentralizovaný kruh

Porovnanie *prežijú aktívne procesory s menšími číslami*; tie s väčším číslom prechádzajú do stavu *lost*. Na záver leader oznámi ukončenie výberu.

iniciátor p

$state_p \leftarrow cand$; send $\langle tok, p\rangle$ do $Next_p$;
repeat
receive $\langle tok, q\rangle$;
if $p = q$ then $state_p \leftarrow leader$
else if $q < p$ then
  begin if $state_p = cand$ then $state_p \leftarrow lost$;
  send $\langle tok, q\rangle$ do $next_p$
end
until $state_p = leader$

neiniciátor p

repeat
receive $\langle tok, q\rangle$; send $\langle tok, q\rangle$ do $Next_p$;
if $state_p = sleep$ then $state_p \leftarrow lost$
until $false$

Tvrdenie 3 *Algoritmus Chang-Roberts rieši problém voľby*
počet správ - $\Theta(N^2)$
čas - $O(N)$

Tvrdenie 4 *V prípade, že sú všetky vrcholy iniciátori, použije algoritmus Chang-Roberts* v priemere *iba* $O(NlogN)$ *správ.*

Výber na kruhoch s $O(NlogN)$ správami

Nutným predpokladom sú **FIFO kanály**. Najprv sa vypočíta minimálna identifikácia a potom proceso
s minimálnym číslom oznámi, že je leadrom.

Pôvodne *aktívne* procesy sa stávajú *pasívnymi*, ak porovnaním zistia, že sa nemôžu stať leadrom.

Porovnanie sa robí so susednými *aktívnymi* identifikátormi. Kolo prežíva lokálne minimum.

Idea sa jednoducho realizuje pri **obojsmerných kanáloch**, ako to prezentoval **Franklin**, 1982.

- Na začiatku sú všetci iniciátori *active*, neiniciátori *passive*
  Každý aktívny proces pošle svoju identifikáciu susedom
- nech aktívny vrchol u dostane v, w:
  - ak $min\{v, w\} < u$ tak u sa stane *passive*
  - ak $min\{v, w\} > u$ tak u pošle opäť svoju identifikáciu susedom
  - ak $min\{v, w\} = u$ tak u sa stane *leader*
- pasívne vrcholy iba posúvajú správy

V každom kole sa aspoň polovica aktívnych vrcholov stane pasívnymi. Preto sa pri výpočte posiela
$O(NlogN)$ **správ**.

Výber na kruhoch s $O(NlogN)$ správami - jednosmerná komunikácia

Porovnanie troch identifikátorov z vrcholov $r, q,$ **p** sa realizuje v **najpravejšom vrchole**

$$\rightarrow r \rightarrow q \rightarrow p \rightarrow$$

- ak q má menší identifikátor ako r a p, tak p vie, že q je lokálne minimum a bude s pamätať jeho hodnotu.
- Ak tomu tak nie je, p sa stane pasívnym

Posielané správy:
$\langle$**one**, .$\rangle$ - touto správou sa p dozvie identifikáciu v q

$\langle$**two**, .$\rangle$ - touto správou sa p dozvie identifikáciu v r

$\langle$**smal**, .$\rangle$ - touto správou oznamujeme identifikáciu leadra

Peterson / Dolev-Klawe-Rodeh **z pohľadu p**

inicializácia
$ci_p \leftarrow p$ v tejto premennej si proces uchováva hodnotu toho, čo posiela
$acn_p \leftarrow undef$ sused proti smeru hodinových ručičiek
$win_p \leftarrow undef$ identifikácia víťaza
$state_p \leftarrow active \setminus passive$ udáva stav procesu – iniciátor, neiniciátor

while $win_p = undef$ **do**

aktívny proces

send $\langle one, ci_p \rangle$; receive $\langle one, q \rangle$; $acn_p \leftarrow q$

if $acn_p = ci_p$ **then**

send $\langle smal, acn_p \rangle$; $win_p \leftarrow acn_p$; receive $\langle smal, q \rangle$

else send $\langle two, acn_p \rangle$; receive $\langle two, r \rangle$

if $acn_p < ci_p \wedge acn_p < r$ **then** $state_p \leftarrow leader$

else $state_p \leftarrow lost$

pasívny proces - posúva správy; zapamätá si len oznámené minimum

receive $\langle one, q \rangle$; send $\langle one, q \rangle$;

receive m; send m;

if m je $\langle smal, q \rangle$ **then** $win_p \leftarrow q$

if $p = win_p$ **then** $state_p \leftarrow leader$

else $state_p \leftarrow lost$

Tvrdenie 5 *Algoritmus P/DKR rieši problém voľby na jednosmernom kruhu, pričom po-*
sielaný počet správ je $O(NlogN)$

Dolný odhad - Pachl, Korach, Rotem, 1984

Predpoklady

1. Budeme predpokladať, že **počítame minimum** (ak je známy leader, minimum vy-
   počítame pomocou N správ; ak máme minimum, leadra určíme pomocou N správ)
2. Kruh je **jednosmerný**
3. Procesy **nepoznajú veľkosť** kruhu
4. Kanály sú FIFO (ak nie, lepšie to byť nemôže)
5. **Všetky procesy sú iniciátori** (nekazíme dolný odhad - je to špeciálny prípad)
6. Algoritmy sú **riadené správami** (neobmedzuje to; ak by algoritmus A nebol taký
   prerobíme ho na algoritmus B. Ten ako reakciu na prijatie správy pošle maximálnu
   postupnosť správ, ktoré by mohol poslať bez prijatia ďalšej správy. Až potom príjme
   ďalšiu správu.)

Pojmy a označenia:

- **D** $= \{(s_1, ..., s_k) : s_i \in P$ a $i \neq j \Rightarrow s_i \neq s_j\}$ D je množina postupností identifikátorov
  $len(s)$ označuje dĺžku postupnosti s
  cyklický shift postupnosti s, $cs(s)$, je postupnosť $s's''$, kde $s = s''s'$;
  CS(s) označuje množinu všetkých cyklických shiftov postupnosti s, $|CS(s)| = len(s)$

- Hovoríme, že kruh je **pomenovaný** postupnosťou $s = (s_1, ..., s_N)$, ak sa identifikačné čísla vyskytujú v rovnakom poradí; niekedy mu hovoríme aj ***s-kruh***.
  Ak $t \in CS(s)$, tak sa t-kruh a s-kruh rovnajú

- Každej správe priradíme postupnosť identifikátorov, tzv ***stopu***. Formálne:
  nech m je správa, ktorú posiela procesor p pred prijatím akejkoľvek správy. Potom $stopa(m) = (p)$
  nech m je správa, ktorú posiela procesor p po prijatí správy so stopou $(s_1, ..., s_k)$. Potom $stopa(m) = (s_1, ..., s_k, p)$
  Ak $stopa(m) = s$, tak správe m hovoríme ***s-správa***

- Nech $E \subseteq D$. Hovoríme, že E ***pokrýva D***, ak

  1. E je uzavretá na prefix -$tu \in E \Rightarrow t \in E$
  2. E cyklicky pokrýva D - $\forall s \in D : CS(s) \cap E \neq \emptyset$

- Na E definujeme dve miery zložitosti.
  M(s, E) $= |\{t \in E : t$ je prefix $r \in CS(s)\}|$
  M$_k$(s, E) $= |\{t \in E : t$ je prefix $r \in CS(s), len(t) = k\}|$

A — algoritmus, ktorý počíta minimum. $\mathbf{E_A}$ množinu s-správ posielaných pri vykonávaní A.

Lema 1 *Ak s je podreťazcom t aj u a pri výpočte algoritmu A na t-kruhu sa vyšle s-správa, tak sa s-správ vyšle aj pri výpočte A na u-kruhu.*

Lema 2 *E_A pokrýva D.*

Lema 3 *Pri výpočte na s-kruhu pošle algoritmus A aspoň $M(s, E_A)$ správ.*

Nech I je konečná podmnožina D. Označme

$Per(I)$ množinu permutácií množiny I

$ave_A(I)$ priemerný počet správ použitých pri výpočte A na všetkých kruhoch pomenovaných identifiká-
tormi z I

$wor_A(I)$ najhorší počet správ

Potom podľa predchádzajúcej lemy pre N-prvkovú množinu I platí

1. $ave_A(I) \geq \frac{1}{N!} \sum_{s \in Per(I)} M(s, E_A)$;
2. $wor_A(I) \geq \max_{s \in Per(I)} M(s, E_A)$

Tvrdenie 6 *Priemerý počet správ algoritmu na hľadanie najmenšieho identifikátora v jednosmernom kru hu je aspoň $N \cdot H_N$*

Dôsledok 1 *Každý decentralizovaný vlnový algoritmus použije aspoň $\Omega(NlogN)$ správ.*

Výber na všeobecných sieťach

Predpoklady - neznáma topológia a neznámi susedia

Veta 1 *Ľubovoľný porovnávací algoritmus výberu na ľubovoľnej sieti používa aspoň* $\mathbf{\Omega(|E| + NlogN)}$ *správ.*

Dôsledok 2 *Decentralizovaný vlnový algoritmus na všeobecných sieťach bez znalosti su*
sedov používa aspoň $\mathbf{\Omega(|E| + NlogN)}$ *správ.*

Aplikácia centralizovaných vĺn

- každý iniciátor spustí vlastnú vlnu A
- správam pridáme značku vlny, do ktorej patria
- keď sa dve vlny stretnú, prežija tá s menším iniciátorom

Algoritmus ECHO(centralizovaná vlna):

- iniciátor pošle správu všetkým susedom
- neiniciátor posiela správu všetkým susedom okrem otca (otec neiniciátora je ten sused, od ktorého sme dostali prvú správu)
- keď neiniciátor dostane správu od všetkých susedov, pošle správu otcovi
- keď iniciátor dostane správu od všetkých susedov, rozhodne

caw_p označuje aktuálnu vlnu; iniciátori - $caw_p := p$, ostatní - $caw_p := undef$

Algoritmus Ex - ECHO so zánikom (decentralizovaný výber)

- každý iniciátor začne svoju vlnu, pričom k správam pridá svoju identifikáciu
- v každom okamihu sa ľubovoľný vrchol zúčastňuje nanajvýš jednej vlny
- situácia vo vrchole p, ktorý patrí do vlny q a je zasiahnutý vlnou r
  - ak $r < caw_p$ tak p prejde do vlny r; $caw_p := r$
  - ak $r > caw_p$ tak p ostane vo vlne caw_p
  - ak $r = caw_p$ tak sa prichádzajúca správa spracuje podľa ECHO algoritmu
- ak vlna caw_p realizuje *decide* vo vrchole p, p sa stane leadrom

Veta 2 *Ak A je centralizovaný vlnový algoritmus, ktorý posiela M správ, tak algoritmus $Ex(A)$ vyberie leadra poslaním maximálne $N \cdot M$ správ.*

Algoritmus Gallager-Humblet-Spira, výber minimálnou kostrou

$C_E \leq C_T + O(|V|)$ (tree algoritmus)
$C_T \leq C_E + O(|E|)$ (echo algoritmus)
$C_T, C_E = \Omega(Nlogn + |E|)$ (Veta.1)
$C_T, C_E = \Theta(Nlogn + |E|)$ (Algoritmus G-H-S(bude nasledovať))

MST -minimálna kostra grafu $G = (V, E)$

Fakt 1 *Ak sú všetky váhy rôzne, je MST určená jednoznačne.*

V sekvenčnom prípade máme dva základné algoritmy, oba založené na greedy metóde.

$F = \emptyset$
kým F nie je kostra, pridaj $\begin{cases} Prim, & \text{minimálnu hranu odchádzajúcu z } F; \\ Kruskal, & \text{minimálnu hranu takú, že } F \text{ tvorí les.} \end{cases}$

Algoritmus Gallager-Humblet-Spira je založený na distribuovanej verzii Kruskalovho algoritmu.

- na začiatku tvorí každý vrchol vlastný fragment
- vrcholy fragmentu F spoločne hľadajú minimálnu odchádzajúcu hranu e_F
- po nájdení e_F je "opačný koniec" hrany e_F vyzvaný na spoluprácu pri spájaní týchto fragmentov
- algoritmus terminuje, keď ostane jediný fragment

Z hľadiska efektívnej implementácie sa treba sústrediť na

— **identifikáciu minimálnej hrany**

— **realizáciu spájania fragmentov**.

Meno fragmentu - kvôli identifikácii toho, či je hrana odchádzajúcou z fragmentu, bu-
dú procesy poznať meno fragmentu FN, v ktorom sa nachádzajú.

Spájanie fragmentov. Budeme si udržiavať informáciu o úrovni L fragmentu. Na za-
čiatku majú všetky fragmenty úroveň 0.

Ak spojíme dva fragmenty s rôznou úrovňou, nový fragment bude mať meno i úroveň
toho z nich, kto mal pôvodne väčšiu úroveň.

V prípade spájania fragmentov rovnakej úrovne sa úroveň zväčší o jedna. Novým
menom sa stane váha hrany, ktorou sa tieto dva fragmenty spojili. Budeme jej hovorit
jadro(core).

Fragmenty $F = (FN, L), F' = (FN', L')$ sa spoja nasledovne:

$L < L' \wedge F \xrightarrow{e_F} F' : \quad F \cup F' = (FN', L')$

$L > L' \wedge F' \xrightarrow{e_F} F : \quad F \cup F' = (FN, L)$

$L = L' \wedge e_F = e_{F'} : \quad F \cup F' = (\omega(e_F), L + 1)$

Lema 4 *Pri dodržaní vyššie uvedených pravidiel proces zmení svoje meno alebo úroveň
nanajvýš $NlogN$ krát.*

Informácie vo vrchole:

- stav: sleep, find, found
- stav hrán: basic, branch, reject
- meno a úroveň fragmentu
- otec smerom k jadru

Inicializácia:
$level_u := 0$, minimálna odchádzajúca hrana uv - branch; ostatné - basic.
send $\langle \mathbf{connect}, 0 \rangle$ do v.

Spojenie fragmentov:
Nech fragmenty $F = (FN, L), F' = (FN', L')$ sú spojené hranou uv

- $L < L'$: send $\langle \mathbf{inic}, L', FN', \frac{find}{found} \rangle$ do u, forward cez F. $F \cup F'$ dedí jadrovú hranu F'
- $L = L'$: v oboch smeroch send $\langle inic, L+1, \omega(uv), \mathbf{find} \rangle$; $F \cup F'$ má jadrovú hranu uv

Výpočet najmenšej odchádzajúcej hrany:

Keď vrchol u dostane správu $\langle \textbf{inic}, L, FN, find \rangle$, skontroluje (v rastúcom poradí vzhľa-
dom na váhu), či niektorá z jeho basic hrán uv nie je odchádzajúcou

1. u pošle $\langle \textbf{test}, L, FN \rangle$ do v
2. ak je v vo fragmente FN, v odpovedá $\langle \textbf{reject} \rangle$, a potom u aj v zaradia
   uv do reject;
3. inak, ak $L \leq level_v, v$ odpovie $\langle \textbf{accept} \rangle$
4. ak $L > level_v, v$ pozdrží spracovanie prichádzajúcej$\langle \textbf{test} \rangle$ správy;
5. ak samotné v čaká na reakciu na testovaciu správu, ktorú poslalo u, v
   neodpovedá

V prípade 4. môžu byť u aj v v tom istom fragmente keď správa $\langle \textbf{inic}, L', FN', \frac{find}{found} \rangle$ putuje do v. Pozdržaním odpovede
nespôsobíme deadlock, pretože vždy existuje fragment minimálnej úrovne, ktorý hľadá odchádzajúcu hranu. V okamihu
keď niektorá z basic hrán akceptuje alebo naopak všetky basic hrany zamietajú, u skončí s hľadaním.

odpovedanie vrcholom jadra (core nodes):

- u čaká na odpoveď od všetkých -*s výnimkou otca* - branch hrán
- u si nastaví stav $found$
- u vypočíta minimálnu váhu ω z týchto odpovedí a jeho minimálnej odchádzajúce hrany (alebo nastaví ∞, ak sa odchádzajúca hrana nenašla)
- u si uloží hranu s minimálnou váhou (či už z poslaných, alebo vlastnú odchádzajúcu)
- u informuje svojho otca poslaním správy $\langle \mathbf{report}, \omega \rangle$

Vrcholy jadra dostanú odpovede (reporty) od *všetkých* svojich susedov, vrátane otca

- ak je doručená hodnota minimálnej hrany ∞, jadro neterminuje
- ak je doručená hodnota minimálnej hrany $< \infty$, jeden z vrcholov jadra pošle smerom k tejto hrane správu $\langle \mathbf{changeroot} \rangle$

Po prijatí správy $\langle \mathbf{changeroot} \rangle$ vrcholom u, ktorý posielal hodnotu najmenšej odchádzajúcej hrany si u nastaví túto hranu ako *branch* a pošle jej správu $\langle \mathbf{connect}, level_u \rangle$

Spojenie dvoch fragmentov:

Ak v príjme správu $\langle \mathbf{connect}, level_u \rangle$ od u, tak $level_v \geq level_u$. Presnejšie, v poslalo $\langle \mathbf{accept} \rangle$ do u.

- ak $level_v > level_u$, send $\langle \mathbf{inic}, level_v, name_v, \frac{find}{found} \rangle$ do u
- ak $level_u = level_v$ a vu nie je branch v, v pozdrží spracovanie **connect** správy
- ak $level_u = level_v$ a vu je branch v,(teda v poslalo do u správu $\langle \mathbf{connect}, level_v \rangle$), u pošle $\langle inic, level_v + 1, \omega(uv), \mathbf{find} \rangle$ do u

Veta 3 *Algoritmus Gallager-Humblet-Spira počíta minimálnu kostru, pričom použije maximálne* $\mathbf{5NlogN + 2|E|}$ *správ*

Algoritmus Korach-Kutten-Moran, výber vychádzajúci z traverzovania

Iniciátor výberu spustí traverzovací algoritmus, pričom si označí svoj token. Pri stretnut rôznych traverzovaní niektoré zanikne. K rozhodnutiu dôjde len v jednom traverzovaní iniciátor tohto traverzovania sa stane leadrom.

Pridáme tokenom informáciu o úrovni. Pri stretnutí dvoch tokenov rovnakej úrovne vznikne token, ktorého úroveň je o jedna väčšia. Ak token s menšou úrovňou vojde do vrchola, v ktorom je alebo bol token s vyššou úrovňou, zaniká.

Aby sa tokeny rovnakej úrovne mohli porovnať, **musia sa stretnúť** v jednom procese. Realizácia- pomocou stavov $wait, annex, chase$

lev_p	: úroveň procesu
cat_p	: iniciátor posledného annex tokena, forwardovaného procesom p (cat = currently active traversal of p)
$wait_p$	: úroveň tokena, ktorý čaká v p
$last_p$	: info o susedovi, do kt. p naposledy forwardovalo $annex$ token úrovne lev
token (q, l)	q je iniciátor tokena, l úroveň tokena

token v stave annex - $\langle annex, q, l\rangle$:

Token (q, l) v stave *anex* vykonáva traverzovací algoritmus až do okamihu, keď nastane jedna z nasledujúcich udalostí:

1. Traverzovací algoritmus úspešne terminoval; q sa stáva leadrom

2. Token prišiel do vrchola $p, level_p > l$. V tomto prípade token zaniká.

3. Token prišiel do vrchola p, v ktorom čaká token úrovne l. Oba tokeny zanikajú. Začne sa nové traverzovanie (T)
   (T): $\begin{cases} wait_p \leftarrow undef; lev_p \leftarrow lev_p + 1; last_p \leftarrow trav(p, lev_p); cat_p \leftarrow p; \\ \text{send } \langle \textbf{annex}, p, lev_p\rangle \text{ do } last_p. \end{cases}$

4. Token prišiel do vrchola p úrovne l, ktorý bol naposledy navštívený tokenom s $cat_p > q$ alebo v stave *chase*. Vtedy token ostane čakať vo vrchole p

5. Token prišiel do vrchola p úrovne l, ktorý bol naposledy navštívený tokenom v stave *annex*, s $cat_p < q$. Token prejde do stavu *chase* a postupuje rovnako ako posledný token (pokúsi sa ho dohnať).
   send $\langle \textbf{chase}, q, l\rangle$ do $last_p; last_p \leftarrow undef$

token v stave chase - $\langle chase, q, l \rangle$:

Token postupuje rovnako ako posledne prechádzajúci token až do okamihu, keď nastane jedna z možností

1. Token prišiel do vrchola $p, level_p > l$. V tomto prípade token zaniká.

2. Token prišiel do vrchola p, v ktorom sa nachádza čakajúci token úrovne l. Oba tokeny zanikajú a vznikne nové traverzovanie (T)

3. Token prišiel do vrchola p úrovne l, ktorý bol naposledy navštívený tokenom v stave *chase*. Token ostane čakať v procese p
   $wait_p \leftarrow q$

čakajúci token - (q,l) čaká na jednu z udalostí:

1. Príchod tokena s vyššou úrovňou. V tomto prípade čakajúci token zaniká.

2. Píchod tokena s rovnakou úrovňou. Oba tokeny zanikajú a naštartuje sa nové traverzovanie (T).

Lema 5 *Ak sa na úrovni l vygenerovalo $k > 1$ tokenov, na úrovni $l+1$ sa vygeneruje aspoň jeden a nanajvýš $k/2$ tokenov.*

Dôkaz:
Pokles na polovicu - token úrovne $l+1$ vzniká spolu so zánikom dvoch tokenov úrovne l

Že bude vygenerovaný aspoň jeden token - *ak v systéme existujú aspoň dva tokeny maximálnej úrovne l, tieto sa v priebehu výpočtu stretnú.*

Veta 4 *Algoritmus Knuth-Kutten-Moran je algoritmom výberu.*

Dôkaz: Ak aspoň jeden proces inicializuje spustenie algoritmu KKM, tak na základe prechádzajúcej lemy sa v (maximálnej) úrovni l vygeneruje iba jeden token, napr. (q, l). Tento token úspešne ukončí traverzovanie. Keďže žiaden z tokenov menšej úrovne nemohol ukončiť traverzovanie, bude v systéme jediný leader - q.

Hovoríme, že algoritmus je **f-traverzovací** pre triedu sietí, ak

1. je traverzovací pre túto triedu
2. v každom výpočte je po $f(x)$ krokoch/posunoch tokena navštívených aspoň $min\{N, x+1\}$ procesov.

Hovoríme, že funkcia f je **konvexná**, ak $f(a)+f(b) < f(a+b)$.

Veta 5 *Nech algoritmus KKM použije $f(x)$-traverzovací algoritmus, kde f je konvexn funkcia. Nech výpočet je iniciovaný k procesmi. Potom sa pri voľbe leadra pošle nanajvý $(1+logk)[f(N)+N]$ správ.*

Keďže pre kruh máme x-traverzovací algoritmus, je počet správ pri **voľbe leadra na kruhu** algoritmom KKM $2NlogN$.

Keďže pre torus máme x-traverzovací algoritmus, je počet správ pri **voľbe leadra na toruse** algoritmom KKM $2NlogN$.

Ak má hyperkocka zmysel pre orientáciu, máme x-traverzovací algoritmus, takže na **výber leadra na hyperkocke** stačí algoritmom KKM $2NlogN$ správ.