Distribuované algoritmy

Distribuované algoritmy

Distribuované systémy:

- skládají se s mnoha paralelně běžících **procesů**
- tyto procesy jsou vzájemně propojeny pomocí **sítě**
- komunikují spolu posíláním **zpráv** přes síťová spojení

Při návrhu a analýze algoritmů pro distribuované systémy je často hlavní pozornost soustředěna na otázky týkající se vzájemné komunikace mezi procesy, synchronizace mezi nimi, množství posílaných zpráv, apod.

Obecně je možné distribuované systémy dělit podle řady různých hledisek.

Jedno z hlavních rozdělení:

- **synchronní**
- **asynchronní**

Synchronní distribuované algoritmy

Popíšeme si jednu z možných variant **synchronních systémů**.

Struktura **komunikační sítě** může být popsána jako orientovaný graf $G = (V, E)$:

- vrcholy grafu představují uzly této sítě
- hrany představují komunikační linky mezi těmito uzly, přes které je možné posílat zprávy

(Typicky se předpokládá, že tento graf je silně souvislý, tj. že existuje cesta z každého vrcholu do každého.)

Je specifikována **množina zpráv** M, které lze přes tyto linky posílat.

- Speciální hodnota *null* (kde $null \notin M$) označuje absenci posílané zprávy, tj. že v daném okamžiku není posílána žádná zpráva.
- V jednom okamžiku je každá z linek z množiny E schopna přenášet jeden prvek z množiny $M \cup \{null\}$.

Synchronní distribuované algoritmy

S každým vrcholem $i \in V$ je spojen jeden **proces**, označený P_i, skládající se z následujících komponent:

- $States_i$ — množina **stavů** (může být nekonečná)
- $Init_i$ — neprázdná podmnožina množiny $States_i$, počáteční stavy
- $msgs_i$ — funkce typu $States_i \times Out\text{-}Links_i \to M \cup \{null\}$,
  — reprezentuje generování zpráv poslaných daným procesem po jednotlivých linkách

  ($Out\text{-}Links_i$ je množina hran vedoucích z vrcholu i)
- $trans_i$ — funkce přiřazující stavům z množiny $States_i$ a vektorům (indexovaných prvky množiny $In\text{-}Links_i$) hodnot z $M \cup \{null\}$ prvky z množiny $States_i$
  — reprezentuje přechody mezi stavy procesu na základě aktuálního stavu a zpráv přijatých daným procesem po jednotlivých linkách

  ($In\text{-}Links_i$ je množina hran vedoucích do vrcholu i)

Synchronní distribuované algoritmy

Běh systému začíná v konfiguraci, kde každý proces P_i je v některém ze stavů z množiny $Init_i$.

Všechny procesy neustále opakují následující dva kroky, kde každý z těchto kroků provádějí všechny procesy synchronizovaně najednou:

- Každý proces P_i vygeneruje zprávy dané funkcí $msgs_i$, které pošle na příslušné linky.

  (Můžeme si představovat, že se tyto zprávy zapíšou do příslušných hran grafu.)

- Každý proces P_i změní svůj stav podle funkce $trans_i$ a přijatých zpráv.

  (Můžeme si představovat, že proces přečte tyto zprávy z příslušných hran grafu, a že poté jsou tyto zprávy z těchto hran smazány.)

Jednomu provedení těchto dvou kroků budeme říkat jedno **kolo** (**round**) posílání zpráv.

Synchronní distribuované algoritmy

- Obecně zde neklademe žádné omezení na to, jaké množství výpočtů musí daný proces P_i provést, aby spočítal hodnoty funkcí $msgs_i$ a $trans_i$.
- Tyto systémy, tak jak byly popsány, jsou plně **deterministické** — pro dané počáteční stavy jednotlivých procesů existuje jediný možný průběh výpočtu.
- Obecně bychom mohli uvažovat i **nedeterministické** systémy, kde by $msgs_i$ a $trans_i$ nebyly funkce, ale relace.

  Případně bychom mohli uvažovat i další možná zobecnění, např. **randomizované** systémy, kde by možné přechody mezi stavy a generované zprávy byly vybírány náhodně podle nějakého pravděpodobnostního rozdělení.

Synchronní distribuované algoritmy

- **Zastavení** procesu P_i je možné modelovat pomocí nějakého speciálního stavu *halt*, kdy proces v tomto stavu již nebude posílat žádné zprávy, přijímané zprávy bude ignorovat a bude už neustále zůstávat v tomto stavu.

- Standardně můžeme předpokládat, že všechny procesy startují najednou na začátku výpočtu.

  Pokud bychom chtěli modelovat to, že procesy startují postupně v různých kolech během výpočtu, můžeme to modelovat například tak, že procesy jsou na začátku v nějakém speciálním stavu *inactive*.

  Součástí grafu bude speciální vrchol s procesem reprezentujícím **prostředí** (**environment**), který v určitých časech bude jednotlivým procesům posílat speciální zprávy *wakeup*, kterými je převede ze stavu *inactive* do aktivního stavu.

Synchronní distribuované algoritmy

Specifickou podoblastí jsou algoritmy pro distribuované systémy, u kterých se počítá s tím, že bude docházet k různým druhům chyb a výpadků, např.:

- ztrátám posílaných zpráv
- chybám při přenosu zpráv, kdy je přijata jiná zpráva, než byla odeslána
- nečekanému zastavení některých procesů
- chybnému chování některých procesů

Výše uvedený model je možné rozšířit o modelování různý druhů těchto chyb.

Synchronní distribuované algoritmy

Formálně můžeme běh daného synchronního distribuovaného systému popsat například jako posloupnost:

$$C_0, \mu_1, \nu_1, C_1, \mu_2, \nu_2, C_2, \ldots$$

kde:

- C_r — reprezentuje stavy jednotlivých procesů po r kolech výpočtu, tj. C_r je funkce, která každému vrcholu $i \in V$ přiřazuje stav z množiny $States_i$.
- μ_r — reprezentuje zprávy poslané jednotlivými procesy v kole r, tj. μ_r je funkce, která každé hraně z množiny E přiřazuje hodnotu z množiny $M \cup \{null\}$
- ν_r — reprezentuje zprávy přijaté jednotlivými procesy v kole r, tj. ν_r je funkce, která každé hraně z množiny E přiřazuje hodnotu z množiny $M \cup \{null\}$

  — ν_r se může lišit od μ_r, pokud modelujeme chyby při přenosu (ztráty zpráv, apod.)

Synchronní distribuované algoritmy

Co se týká **vstupu** a **výstupu**, předpokládá se, že:

- jednotlivé části vstupu jsou nějakým způsobem uloženy v počátečních stavech procesů,
- jednotlivé části výstupu budou poté, co výpočet skončí, nějakým způsobem uloženy ve stavech jednotlivých procesů.

Poznámka: U distribuovaných systémů nemusí být vždy úplně snadné rozpoznat, kdy výpočet končí.

U některých algoritmů například může být na závěr výpočtu provedena nějaká závěrečná fáze, kdy jsou jednotlivé procesy rozesláním příslušných zpráv informovány o tom, že výpočet skončil.

Synchronní distribuované algoritmy

Z hlediska složitosti se u synchronních distribuovaných systémů zkoumají především:

- **časová složitost** — doba výpočtu se většinou počítá jako počet provedených kol posílání zpráv
- **komunikační složitost** — většinou se počítá jako celkový počet zpráv poslaných během výpočtu

  Někdy se bere v úvahu i velikost těchto zpráv, tj. celkový počet bitů ve všech poslaných zprávách.

Leader election

Ukážeme si několik příkladů distribuovaných algoritmů.

Budou řešit problém, kdy je mezi všemi procesy třeba zvolit jeden, který pak například provede nějakou akci.

Tento problém se nazývá **leader election**.

Poznámka: Řešení tohoto problému bude typicky sloužit jako součást nějakého většího algoritmu.

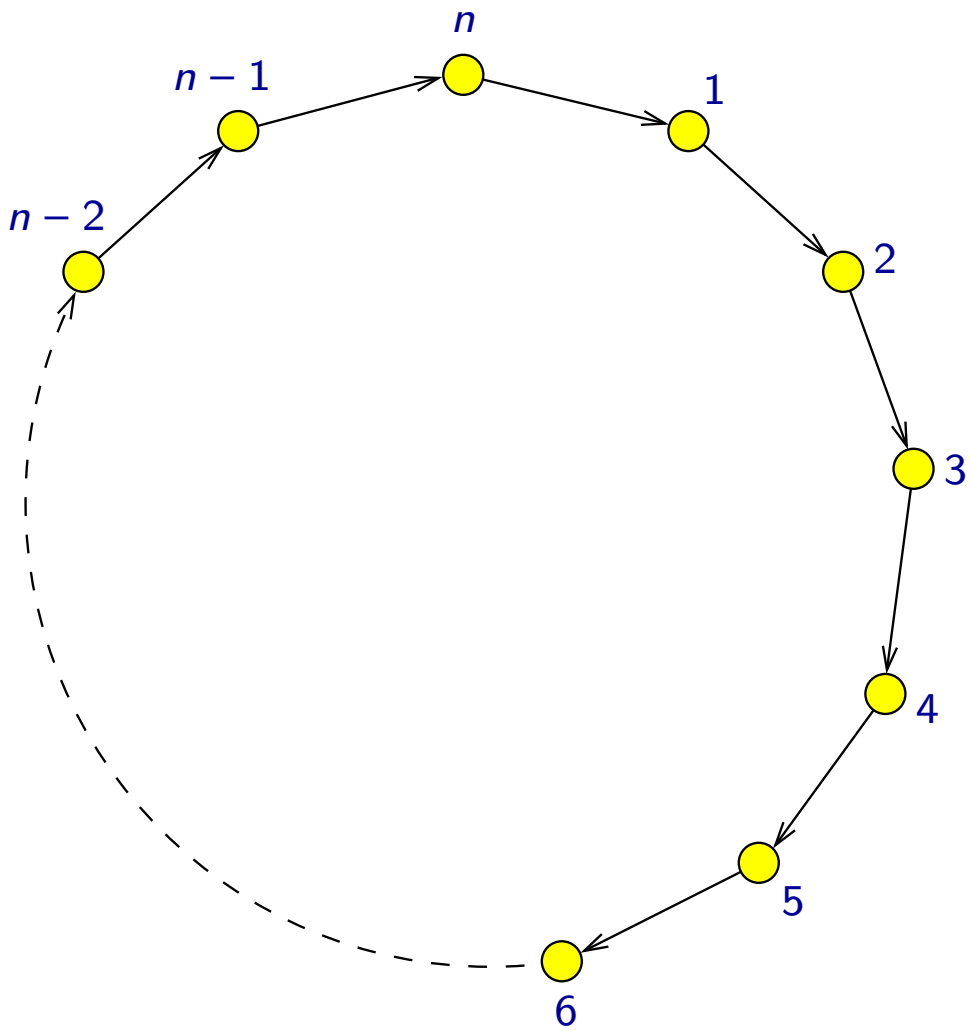
Nejedná se o typický vstupně-výstupní problém, kde by bylo určeno, co je vstupem a výstupem.

Cílem je, aby po skončení algoritmu právě jeden z procesů přešel do nějakého speciálního stavu, označeného např. LEADER.

Ostatní procesy pak může tento zvolený proces informovat o svém zvolení rozesláním příslušných zpráv.

Leader election

Zaměřme se nejprve na jednodušší případ, kdy má síť podobu (jednosměrného) **kruhu**:



Leader election

- Vrcholy kruhu označme čísly $1, 2, \ldots, n$.
- Při počítání s indexy vrcholů budeme ztotožňovat vrcholy s indexy, které se rovnají modulo n.

  Například $i + 1$ bude v případě, že $i = n$, označovat vrchol 1.
  Podobně, $i - 1$ bude pro $i = 1$ označovat vrchol n.
- Předpokládejme ovšem, že jednotlivé procesy neznají svou pozici na kruhu, tj. neví, na kterém vrcholu se nachází, ani neznají celkový počet procesů n.
- Na druhou stranu předpokládejme, že každý proces ví:
  - kam má poslat zprávu, kterou chce poslat svému následníkovi na kruhu
  - odkud má přijímat zprávy od svého předchůdce na kruhu

  Tj. že každý proces má nějak lokálně pojmenované hrany, které do něj a z něj vedou, nezná ale čísla vrcholů, ze kterých a kam tyto hrany vedou.

Leader election

Není těžké si rozmyslet, že pokud by všechny procesy byly naprosto identické a začínaly by všechny v tomtéž počátečním stavu, tak deterministickým distribuovaným algoritmem není možné tento problém řešit:

- Indukcí podle počtu provedených kol posílání zpráv můžeme ukázat, že pro každé $r \in \mathbb{N}$ platí, že po provedení r kol by všechny procesy byly ve stejném stavu.

  (Deterministické procesy, které jsou ve stejném stavu a dostanou stejné zprávy, přejdou do téhož stavu.)

Potřebujeme tedy nějaký způsob, jak **rozbít symetrii** (**symmetry breaking**).

Leader election

Budeme předpokládat, že každý proces má přiděleno **unikátní ID** (**UID**):

- Hodnoty UID jsou prvky z nějaké dostatečně velké **lineárně uspořádané** množiny, např. z množiny všech přirozených čísel $\mathbb{N}$.

  Tj. pro libovolná dvě UID je možné určit, které z nich je větší a které menší, nebo případně, že se rovnají.

- Tato UID jsou unikátní v tom smyslu, že dva různé procesy nikdy nebudou mít přiděleno totéž UID.

- Hodnoty UID nejsou v žádném vztahu k indexům vrcholů na kruhu a nemusí ani tvořit posloupnost po sobě jdoucích čísel.

- Každý proces zná své UID.

- S hodnotami UID mohou procesy pracovat stejně jako s jakýmikoli jinými hodnotami (např. ukládat je do paměti, posílat ve zprávách, apod.)

Leader election — algoritmus LCR

První algoritmus je jednoduché přímočaré řešení — označme tento algoritmus jako **algoritmus LCR** podle jmen autorů (Le Lann, Chang, Roberts).

Neformální popis:

- Každý proces pošle své UID podél kruhu.
- Když proces přijme zprávu s nějakým UID, porovná toto UID se svým UID:
  - Pokud je toto UID větší než jeho vlastní, přepošle jej dál.
  - Pokud je toto UID menší než jeho vlastní, nepřeposílá ho dál.
  - Pokud je toto UID stejné jako jeho vlastní, prohlásí se daný proces zvoleným leaderem.

Leader election — algoritmus LCR

Formální popis:

- Množina zpráv M: množina všech UID
- Pro každé $i \in \{1, 2, \ldots, n\}$ jsou stavy z množiny $States_i$ dány hodnotami následujících tří proměnných:
  - u — typu UID
  - *send* — bude obsahovat hodnoty typu UID nebo *null*
  - *status* — bude nabývat hodnot z množiny $\{\text{UNKNOWN}, \text{LEADER}\}$
- Pro každé $i \in \{1, 2, \ldots, n\}$ množina $Init_i$ obsahuje jediný počáteční stav, daný následujícími počátečními hodnotami proměnných:
  - u — hodnota UID procesu na vrcholu i
  - *send* — stejná hodnota, jako je počáteční hodnota proměnné u, tj. UID procesu na vrcholu i
  - *status* — na začátku inicializováno na hodnotu UNKNOWN

Leader election — algoritmus LCR

- Pro každé $i \in \{1, 2, \ldots, n\}$ je funkce $msgs_i$ popisující generování zpráv dána následovně:
  - pošli aktuální hodnotu proměnné *send* procesu $i + 1$
- Pro každé $i \in \{1, 2, \ldots, n\}$ je funkce $trans_i$ popisující změny stavu procesu dána následujícím pseudokódem:

---

send := *null*
if přišla zpráva obsahující UID v **then**
  case
    $v > u$: *send* := v
    $v = u$: *status* := LEADER
    $v < u$: nedělej nic

Leader election — algoritmus LCR

Řekněme, že i_{max} je index procesu s největším UID.

Není těžké ověřit, že:

- Po n kolech nastaví proces i_{max} hodnotu své proměnné *status* na LEADER.

  (Jedině zprávám s UID procesu i_{max} se podaří „obkroužit“ celý kruh.)

- Žádný jiný proces nebude mít nastavenu hodnotu proměnné *status* na LEADER.

Po n kolech tedy bude vybrán leader.

Následně by mohl tento vybraný proces poslat zprávu s touto informací všem procesům.

Časová složitost algoritmu LCR je tedy $\Theta(n)$.

Leader election — algoritmus LCR

Je také zřejmé, že celkový počet poslaných zpráv je $\mathcal{O}(n^2)$.

V nejhorším případě může být počet poslaných zpráv $\Omega(n^2)$.

Komunikační složitost je tedy $\Theta(n^2)$.

Algoritmus LCR má časovou složitost $\Theta(n)$ a komunikační složitost $\Theta(n^2)$.

Leader election — algoritmus HS

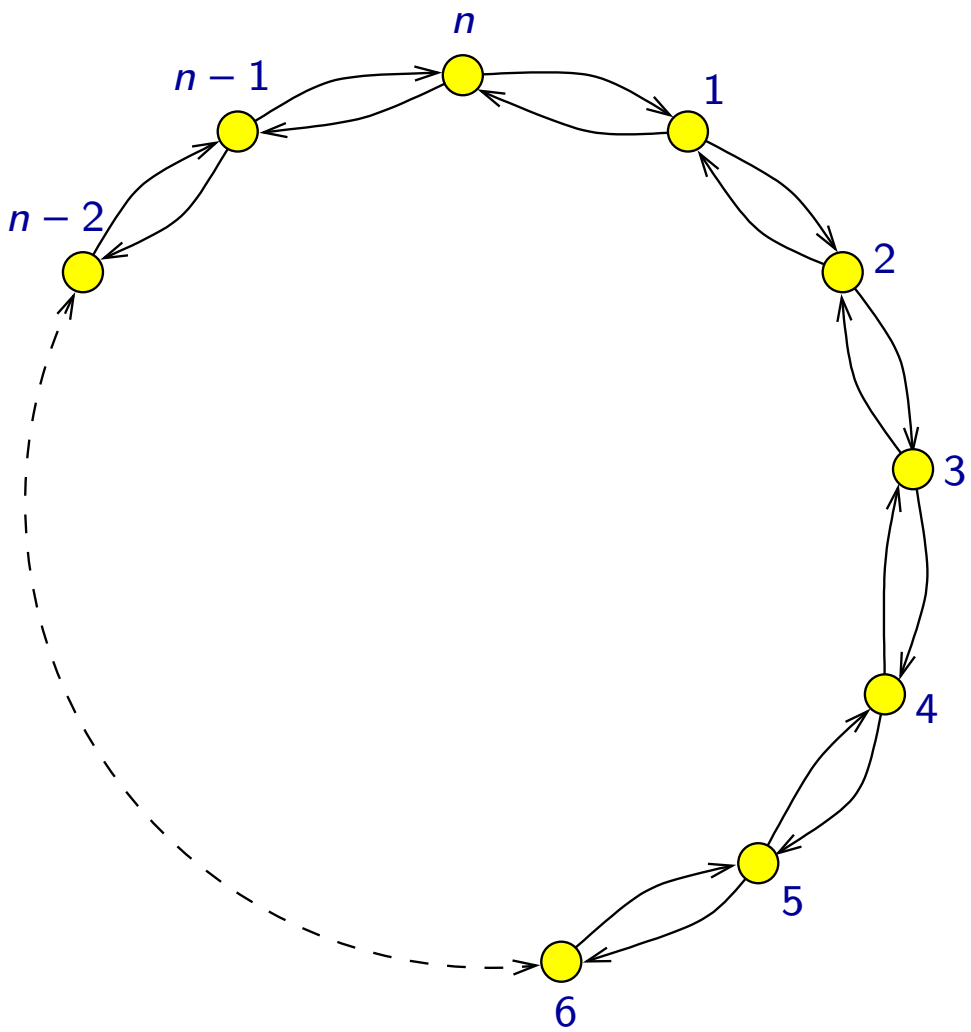
Ukážeme si algoritmus řešící problém leader election s asymptoticky stejnou časovou složitostí jako algoritmus LCR (tj. $\Theta(n)$), ale s komunikační složitostí $\mathcal{O}(n \log n)$.

Označme tento algoritmus podle jmen autorů jako **algoritmus HS** (Hirschberg, Sinclair).

- Algoritmus HS, podobně jako algoritmus LCR, předpokládá, že struktura sítě má podobu **kruhu**.
- Na rozdíl od algoritmu LCR tento kruh ale není jednosměrný, nýbrž **obousměrný**.
- Proces s indexem i tedy může posílat zprávy i přijímat zprávy od procesů s indexy $i - 1$ a $i + 1$.

Leader election — algoritmus HS

Obousměrný kruh:

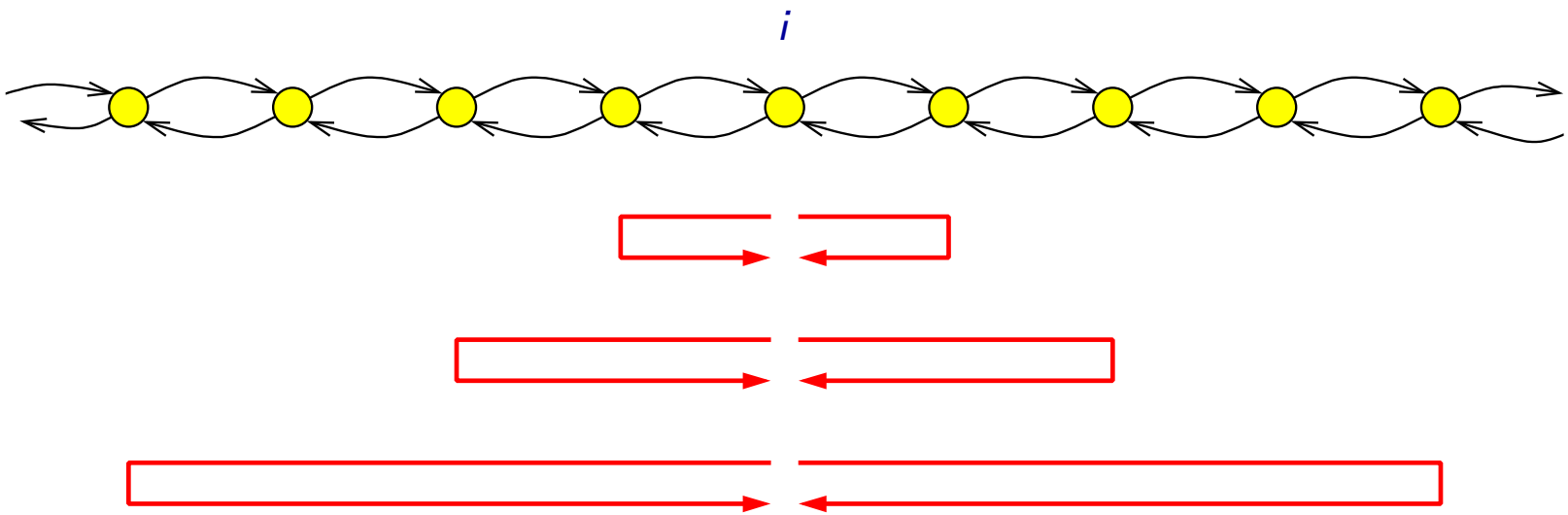


Leader election — algoritmus HS

Neformální popis algoritmu:

- Všechny procesy pracují ve „fázích“ $0, 1, 2, 3, \ldots$
- Ve fázi ℓ vyšle proces i do obou směrů „tokeny“ obsahující jeho UID.

  Tyto tokeny by měly cestovat do vzdálenosti 2^ℓ a pak se vrátit zpět k procesu i.



Leader election — algoritmus HS

- Pokud se oba tokeny úspěšně vrátí, pokračuje proces i další fází, tj. fází $\ell + 1$.
- Token se ovšem nemusí vrátit.

  Každý proces j, který leží na cestě tokenu, porovná své UID u_j s UID u_i v daném tokenu:

  - Pokud $u_i < u_j$, proces j daný token „zahodí“ a nebude ho přeposílat dál.
  - Pokud $u_i > u_j$, proces j daný token přepošle dál.
  - Pokud $u_i = u_j$, znamená to, že daný token stihl „obkroužit“ celý kruh ještě předtím, než se obrátil na cestu zpět.

    V tomto případě je proces j vybrán jako leader.

Poznámka: Stejně jako v případě algoritmu LCR je vybrán jako leader proces s největším UID.

Leader election — algoritmus HS

Formální popis:

- Množina zpráv M je tvořena trojicemi tvaru

  $$(u, \mathit{flag}, \mathit{hop\text{-}count})$$

  kde:

  - u — UID
  - flag — hodnota z množiny $\{\,\text{OUT}, \text{IN}\,\}$
  - $\mathit{hop\text{-}count}$ — kladné celé číslo

- Pro každé i je množina stavů $States_i$ tvořena stavy určenými hodnotami následujících proměnných:

  - u — typu UID
  - $send_+$, $send_-$ — obsahují buď prvek z množiny M nebo $null$
  - $status$ — bude obsahovat hodnoty z množiny

    $$\{\text{UNKNOWN}, \text{LEADER}\}$$

  - $phase$ — bude obsahovat přirozené číslo

Leader election — algoritmus HS

- Pro každé i bude množina počátečních stavů $Init_i$ obsahovat jediný stav daný následujícími počátečními hodnotami proměnných:
  - u — UID procesu ve vrcholu i
  - $send_+$ — trojice $(u, \text{OUT}, 1)$, kde u je UID procesu ve vrcholu i
  - $send_-$ — trojice $(u, \text{OUT}, 1)$, kde u je UID procesu ve vrcholu i
  - $status$ — hodnota UNKNOWN
  - $phase$ — hodnota 0

- Pro každé i je funkce $msgs_i$, popisující generování zpráv, definovaná následovně:
  - pošli aktuální hodnotu proměnné $send_+$ procesu $i + 1$
  - pošli aktuální hodnotu proměnné $send_-$ procesu $i - 1$

Leader election — algoritmus HS

- Pro každé i je přechodová funkce $trans_i$ dána následujícím pseudokódem:

$send_+ := null$
$send_- := null$
if od procesu $i-1$ přišla zpráva (v, OUT, h) **then**
  case
    $v > u$ **and** $h > 1$: $send_+ := (v, \text{OUT}, h-1)$
    $v > u$ **and** $h = 1$: $send_- := (v, \text{IN}, 1)$
    $v = u$: $status := \text{LEADER}$
if od procesu $i+1$ přišla zpráva (v, OUT, h) **then**
  case
    $v > u$ **and** $h > 1$: $send_- := (v, \text{OUT}, h-1)$
    $v > u$ **and** $h = 1$: $send_+ := (v, \text{IN}, 1)$
    $v = u$: $status := \text{LEADER}$

$\vdots$

Leader election — algoritmus HS

$\vdots$

if od procesu $i-1$ přišla zpráva $(v, \text{IN}, 1)$ **and** $v \neq u$ **then**
  $send_+ := (v, \text{IN}, 1)$

if od procesu $i+1$ přišla zpráva $(v, \text{IN}, 1)$ **and** $v \neq u$ **then**
  $send_- := (v, \text{IN}, 1)$

if od procesů $i-1$ a $i+1$ přišla od obou zpráva $(u, \text{IN}, 1)$ **then**
  $phase := phase + 1$
  $send_+ := (u, \text{OUT}, 2^{phase})$
  $send_- := (u, \text{OUT}, 2^{phase})$

Leader election — algoritmus HS

Analýza komunikační složitosti:

- Ve fázi 0 posílají token všechny procesy — celkem je to $4n$ zpráv.
- Ve fázi $\ell > 0$ posílá proces token právě tehdy, když se mu úspěšně vrátily oba tokeny ve fázi $\ell - 1$.

  To nastává právě tehdy, když v obou směrech do vzdálenosti $2^{\ell-1}$ neexistoval žádný proces s vyšším UID.

  To znamená, že v každé skupině $2^{\ell-1} + 1$ po sobě jdoucích procesů bude vždy nejvýše jeden, který ve fázi ℓ začne vysílat tokeny.

  Počet procesů, které ve fázi ℓ začnou vysílat tokeny, je tedy nejvýše

$$\left\lfloor \frac{n}{2^{\ell-1} + 1} \right\rfloor$$

- Ve fázi ℓ putují tokeny nejvýše do vzdálenosti 2^{ℓ}.

Leader election — algoritmus HS

- Celkový počet zpráv poslaných ve fázi ℓ je tedy omezen hodnotou

$$4 \cdot \left(2^{\ell} \cdot \left\lfloor \frac{n}{2^{\ell-1}+1} \right\rfloor \right) \leq 8n$$

- Celkový počet fází předtím, než je zvolen leader, je nejvýše $1 + \lceil \log_2 n \rceil$ (včetně fáze 0).
- Celkový počet poslaných zpráv je tedy nejvýše $8n(1 + \lceil \log_2 n \rceil)$, což je $\mathcal{O}(n \log n)$.

Leader election — algoritmus HS

Analýza časové složitosti:

- Počet kol ve fázi ℓ je $2 \cdot 2^{\ell} = 2^{\ell+1}$.
- Poslední fáze trvá n kol — je to nedokončená fáze.
- Předposlední fáze je fáze $\ell = \lceil \log_2 n \rceil - 1$.
  Doba trvání této fáze je
  $$2^{\lceil \log_2 n \rceil}.$$
- Součet trvání všech fází kromě poslední je tedy nejvýše
  $$2 \cdot 2^{\lceil \log_2 n \rceil}.$$
- Pokud je n mocnina dvojky, bude celkový počet všech fází $3n$.
  Pokud n není mocnina dvojky, bude celkový počet všech fází $5n$.
- Časová složitost je tedy $\mathcal{O}(n)$.

Leader election — algoritmus HS

Algoritmus HS má časovou složitost $\mathcal{O}(n)$ a komunikační složitost $\mathcal{O}(n \log n)$.

Poznámka:

- Dá se dokázat, že jakýkoli algoritmus, který řeší problém leader election na obousměrném kruhu, a který má vlastnost, že s UID provádí pouze operaci porovnání, musí mít komunikační složitost minimálně $\Omega(n \log n)$.

Leader election

Uvažujme algoritmy, kde UID jsou kladná přirozená čísla, se kterými je možné provádět i libovolné jiné operace než jen porovnání.

Za cenu extrémního nárůstu časové složitosti je možné snížit komunikační složitost na $\mathcal{O}(n)$.

Leader election — algoritmus *TimeSlice*

Algoritmus TimeSlice:

- Tento algoritmus vede ke zvolení procesu s nejmenším UID.
- Předpokládá se, že všechny procesy znají celkový počet procesů n.
- Algoritmus pracuje ve fázích $1, 2, 3, \ldots$
- Každá fáze se skládá z n kol.
- Každá fáze v, skládající se z kol

  $$(v-1)n+1,\ (v-1)n+2,\ \ldots\ ,\ v \cdot n,$$

  je věnovaná tomu, že v ní může po kruhu obíhat pouze token s UID v.
- Pokud existuje proces i s UID v a až do kola $(v-1)n+1$ tomuto procesu nepřišla žádná zpráva, prohlásí se tento proces za leadera a nechá kolovat token, kterým informuje všechny ostatní procesy o této volbě.

Leader election — algoritmus *TimeSlice*

Časová složitost algoritmu *TimeSlice* je zhruba $u_{min} \cdot n$, kde u_{min} je hodnota minimálního UID mezi procesy.

Počet poslaných zpráv je n.

Poznámka:
Pokud bychom chtěli místo procesu s minimálním UID zvolit proces s maximálním UID, můžeme postupovat následovně:

- Podobně jako v předchozím případě necháme nejprve zvolit proces s nejmenším UID.
- Tento proces inicializuje kolování tokenu, kdy každý z procesů pošle v daném tokenu maximum z hodnoty UID obsaženém v přijatém tokenu a svého UID.
- Komunikační složitost tohoto algoritmu je stále $\mathcal{O}(n)$.

Leader election — algoritmus *VariableSpeeds*

Algoritmus *TimeSlice* vyžaduje, aby procesy předem znaly celkový počet procesů n.

Následující algoritmus, nazvaný *VariableSpeeds*, tuto informaci nepotřebuje, ale jeho časová složitost je ještě horší.

- Rozhodně se nejedná o algoritmus, který by byl použitelný v praxi.
- Role tohoto algoritmu je v tom, že slouží jako protipříklad toho, že by každý algoritmus řešící problém leader election na kruhu musel mít komunikační složitost nejméně $\Omega(n \log n)$.
- Komunikační složitost algoritmu *VariableSpeeds* je $\mathcal{O}(n)$.

Leader election — algoritmus *VariableSpeeds*

Algoritmus VariableSpeeds:

- Každý proces i inicializuje kolování tokenu, který ponese informaci o UID procesu i (označme toto UID u_i).
- Tokeny se budou pohybovat různou rychlostí.

  Token s UID v se bude pohybovat rychlostí 1 zpráva každých 2^v kol — tj. každý proces, který takový token dostane, bude čekat 2^v kol, než ho pošle dál.
- Mezitím si každý proces udržuje nejmenší UID, které zatím viděl, a zahazuje každý token, který má vyšší UID.
- Jestliže se nějaký token vrátí tomu procesu, který ho vypustil, prohlásí se tento proces za leadera.

Leader election — algoritmus *VariableSpeeds*

Analýza algoritmu VariableSpeeds:

- Označme u_{min} nejmenší UID mezi procesy.
- V okamžiku, kdy token s UID u_{min} oběhl celý kruh:
  - token s druhým nejmenším UID oběhl nejvýše polovinu kruhu
  - token s třetím nejmenším UID oběhl nejvýše čtvrtinu kruhu
  - token se čtvrtým nejmenším UID oběhl nejvýše osminu kruhu
  - ...

  Obecně token s k-tým nejmenším UID oběhl nejvýše část kruhu danou zlomkem

$$\frac{1}{2^{k-1}}$$

- Poslání tokenu s UID u_{min} vyžaduje n zpráv.
- Celkový počet všech poslaných zpráv bude menší než $2n$.

Leader election — algoritmus *VariableSpeeds*

- Cesta tokenu s UID u_{min} vyžaduje poslání n zpráv.
  Poslání každé zprávy trvá $2^{u_{min}}$ kroků.
  Celková doba běhu algoritmu je tedy $n \cdot 2^{u_{min}}$ kol.

Časová složitost algoritmu *VariableSpeeds* je $\mathcal{O}(n \cdot 2^{u_{min}})$ a jeho komunikační složitost je $\mathcal{O}(n)$.

Leader election — algoritmus *FloodMax*

Uvažujme nyní problém leader election na **obecném grafu** $G = (V, E)$.

Budeme předpokládat, že:

- Graf G je silně souvislý — tj. existuje v něm cesta z každého vrcholu u do každého vrcholu v.
- Všechny procesy znají hodnotu *diam*, což je horní odhad pro **průměr** grafu G, tj. takovou hodnotu, že pro každé dva vrcholy $u, v \in V$ platí, že:
  - délka nejkratší cesty z u do v je menší nebo rovna *diam*.

Algoritmus FloodMax — neformální popis:

- Každý proces udržuje informaci o největším UID, které zatím viděl.
- V každém kole toto největší UID posílá všem svým sousedům.
- Po *diam* kolech se proces, jehož UID je stejné jako maximální UID, které zatím viděl, prohlásí za leadera.

Leader election — algoritmus *FloodMax*

Formální popis:

- Množina zpráv M: množina všech UID
- Pro každé i je množina stavů $States_i$ tvořena stavy danými hodnotami následujících proměnných:
  - u — UID procesu i
  - *max-uid* — maximální UID, které proces i zatím viděl
  - *status* — prvek z množiny $\{\text{UNKNOWN}, \text{LEADER}, \text{NON-LEADER}\}$
  - *round* — číslo kola (číslo z množiny $\mathbb{N}$)
- Pro každé i je množina počátečních stavů $Init_i$ tvořena jedním stavem daným následujícími počátečními hodnotami proměnných:
  - u — UID procesu i
  - *max-uid* — UID procesu i
  - *status* — hodnota UNKNOWN
  - *round* — hodnota 0

Leader election — algoritmus *FloodMax*

- Pro každé i je funkce $msgs_i$ specifikující posílané zprávy definována následovně:
  - Pokud $round < diam$, tak všem procesům $j \in V$ takovým, že $(i, j) \in E$, pošli zprávu s UID $max\text{-}uid$.
- Pro každé i je přechodová funkce $trans_i$ popsána následujícím pseudokódem:

$round := round + 1$
nechť U je množina všech UID, které přišly od procesů j, kde $(j, i) \in E$
$max\text{-}uid := \max(\{max\text{-}uid\} \cup U)$
if $round = diam$ **then**
  if $max\text{-}uid = u$ **then** $status :=$ LEADER
  else $status :=$ NON-LEADER

Leader election — algoritmus *FloodMax*

Časová složitost algoritmu *FloodMax* je $\mathcal{O}(diam)$ a jeho komunikační složitost je $\mathcal{O}(diam \cdot |E|)$.

Poznámky:

- Počet posílaných zpráv je možné o něco snížit (i když ne asymptoticky v nejhorším případě) tím, že každý proces bude posílat danou hodnotu *max-id* jen v kolech, kdy se změnila, tj. kdy se o příslušném UID poprvé dozvěděl.
- Existují algoritmy řešící problém leader election bez toho, aby procesy musely znát hodnotu *diam*.