Obsah

1 Distribuované systémy **3**

1.1 Čo je distribuovaný systém ? 3
1.2 Počítačové siete 4
1.2.1 Rozľahlé siete 4
1.2.2 Lokálne siete 5
1.3 Viacprocesorové počítače 5
1.4 Kooperujúce procesy 6

2 Architektúra a jazyky **7**

2.1 Architektúra 7
2.2 OSI referenčný model 7
2.2.1 OSI model v LAN : IEEE štandardy 8
2.3 Podpora jazykmi 8
2.4 Distribuované algoritmy 8
2.4.1 Distribuované versus centralizované algoritmy 9
2.4.2 Konverzácia jednou správou 9
2.4.3 Konverzácia dvomi správami 9
2.4.4 Konverzácia tromi správami 10
2.4.5 Konverzácia štyrmi správami 10
2.4.6 Konverzácia piatimi správami [Belsnes 1976] 11
2.5 Model distribuovaných výpočtov 11
2.5.1 Neformálny popis 11
2.5.2 Formálny popis 11
2.5.3 Systémy s asynchrónnym posielaním správ 12
2.5.4 Systémy so synchrónnym posielaním správ 12
2.5.5 Overovanie vlastností prechodových systémov 13
2.6 Zložitosť 14
2.7 Úlohy 14

3 Distribuovaný výber **15**

3.1 Dvojsmerné kruhy, priemerný prípad. 16
3.1.1 Algoritmus - P 16
3.1.2 Algoritmus-D 17
3.2 Dvojsmerné kruhy, najhorší prípad 18
3.2.1 Algoritmus výberu 18

3.2.2 Analýza algoritmu . . . 19
3.2.3 Modifikácia algoritmu pre všeobecné dvojsmerné kruhy . . . 21
3.3 Jednosmerné kruhy, Najhorší prípad . . . 21
3.3.1 Úpravy algoritmu . . . 22
3.3.2 Algoritmus . . . 22
3.3.3 Korektnosť a zložitosť . . . 23
3.4 Synchrónne kruhy . . . 25
3.5 Topológia všeobecnej siete . . . 25
3.6 Úlohy . . . 25

4 Minimálna kostra **27**
4.1 Sekvenčné algoritmy . . . 27
4.2 Popis algoritmu GHS . . . 27
4.3 Algoritmus GHS . . . 28
4.4 Korektnosť a zložitosť . . . 30

5 Traverzovacie algoritmy **31**
5.1 Čisté traverzovanie . . . 31
5.2 Prehľadávanie do šírky . . . 32
5.3 Prehľadávanie do hĺbky . . . 33
5.4 Traverzovanie torusu . . . 35
5.5 Traverzovanie hyprekociek . . . 36
5.6 Úlohy . . . 36

6 Routovacie algoritmy **37**
6.1 Algoritmus Netchange . . . 37
6.2 Routovanie s kompaktnými routovacími tabuľkami . . . 39
6.2.1 Schéma značenia stromu . . . 39
6.3 Intervalové routovanie . . . 40
6.3.1 Efektívnosť intervalového routovania . . . 41
6.3.2 Optimálnosť intervalového routovania : špeciálne topológie . . . 41
6.4 Prefixové routovanie . . . 43
6.5 Hierarchické routovanie . . . 44
6.6 Úlohy . . . 45

7 Problém dohody v nespoľahlivých systémoch **47**
7.1 Výpočtové modely . . . 47
7.2 Problém dohody . . . 48
7.3 Problém interaktínej konzistencie . . . 49
7.4 Problém generálov . . . 49
7.5 Vzťahy medzi problémami dohody. . . . 49
7.6 Riešiteľnosť problémov dohody . . . 50
7.7 Zložitosť . . . 54
7.7.1 Horné odhady . . . 54
7.7.2 Dolné odhady . . . 55

Kapitola 1

Distribuované systémy

Upozornenie : Obsah tohto dokumentu nemusí na niektorých miestach plne zodpovedať pôvodnému dokumentu, a to z dvoch dôvodov :

1. Pri prepise do súčasnej formy sa mohli vyskytnúť chyby spôsobené ľudskou nepozornosťou.
2. Zopár anglických výrazov buď nemá slovenský ekvivalent, a lebo sa prekladá dosť zložito a krkolomne, okrem toho moja slovná zásoba dosť kolíše podľa dennej doby a obsahu kyslíka vo vzduchu, čo znamená, že som mohol niečo preložiť nie úplne presne.

Z týchto dôvodov preto neodporúčam brať tento dokument ako plne záväzný.

Úvod

Distribuovaný systém - počítačové aplikácie, v ktorých viaceré počítače alebo procesory spolupracujú nejakým spôsobom.

Príklad : Rozľahlé komunikačné siete (WAN), lokálne siete (LAN), viacprocesorové počítače v ktorých má každý procesor vlastnú riadiacu jednotku, systémy kooperujúcich procesov.

1.1 Čo je distribuovaný systém ?

distribuovaný systém - prepojenie sústavy autonómnych počítačov, procesov alebo procesorov

"autonómny" - uzly s vlastným riadením (nie SIMD)

"prepojenie" - uzly si vymienňajú informácie

Motivácia

distribuovaný systém môže byť uprednostnený pred sekvenčným systémom z viacerých dôvodov :

- Výmena informácií (WAN)
- Zdieľanie zdrojov (LAN)
- Zvýšenie spoľahlivosti využitím replikácie
- Zvýšenie výkonu využitím paralelizmu
- Zjednodušenie návrhu využitím špecializácie

1.2 Počítačové siete

WAN a LAN - relevantné rozdiely s ohľadom na vývoj algoritmov :

- Parametre spoľahlivosť
- Komunikačný čas
- Homogénnosť
- Vzájomná dôvera

1.2.1 Rozľahlé siete

Historický vývoj

ARPANET (ARPA - Advanced Research Projects Agency of US Department of Defence) - 1969, prepojenie niekoľko sto uzlov a siete použitím podobnej technológie (MILNET, CYPRESS)

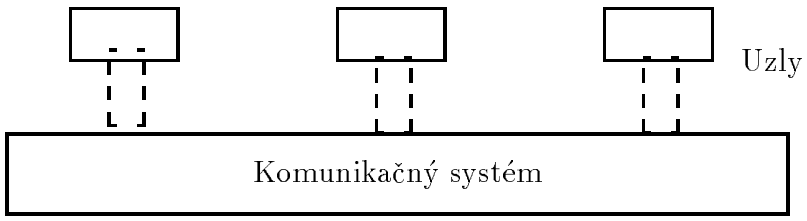
UNIX uucp program (UNIX-to-UNIX CoPy) - výmena informácií cez sieť UUCP

BITNET

Internet - virtuálna sieť, prepája všetky tieto siete (cez brány)
- uvedenie uniformného adresovacieho priestoru (e-mail adresy nevyžadujú žiadne informácie o sieti)

Organizácia

Ethernet *Xerox corporation*
štruktúra zbernicového typu, jedna správa prenášaná naraz (lacné, ale nedá sa zmeniť veľkosť)



Token ring *IBM Zurüch Laboratory*
štruktúra zbernicového typu, viacero správ prenášaných naraz

SNA *IBM* sieť typu point-to-point

Algoritmické problémy

Použitie LAN na distribuované vykonávanie aplikačných programov (procesy sa rozdelia po uzloch siete) vyžaduje :

1. Vysielanie a synchronizáciu
2. Výber
3. Detekciu ukončenia
4. Alokáciu zdrojov
5. Vzájomné vylúčenie
6. Detekciu a riešenie uviaznutia
7. Distribuovanú správu súborov (časové pečiatky)

Organizácia a algoritmické problémy WAN

- sú organizované ako point-to-point siete (komunikácia mechanizmom daným pre dané komunikačné uzly, ako pri telefónnej linke, sklenenom vlákne alebo satelitnom spojení)
- reprezentované ako grafy

Hlavný dôvod WAN je výmena informácií (e-mail, nástenky, vzdialene pristupované súbory, databázy).

Implementácia komunikačného systému vyžaduje :

1. Spoľahlivosť point-to-point výmeny dát (chyby pri prenose)
2. Výber komunikačnej cesty (routing)
3. Riadenie zaplnenosti (priepustnosť komunikačnej siete)
4. Ochrana pred uviaznutím (store-and-forward siete)
5. Bezpečnosť (autentizácia, šifrovanie, vírusy)

1.2.2 Lokálne siete

LAN používajú organizácie na prepojenie počítačov za účelom :

- zdieľania zdrojov (súbory, hardwareové periférie)
- umožnenie výmeny informácií
- urýchlenia výpočtov alebo obnovy po chybe

1.3 Viacprocesorové počítače

Viacprocesorový počítač pozostáva z niekoľkých procesorov na malej ploche. Procesory sú homogénne, geografická vzdialenosť je veľmi malá.

Paralelný počítač - viacprocesorový počítač na zlepšenie rýchlosti výpočtu

Replikačné systémy - viacprocesorový počítač na zvýšenie spoľahlivosti

Paralelný počítač - klasifikácia

SIMD (jedna inštrukcia, viacnásobné dáta) (nehodnotíme ako distribuovaný systém)

MIMD (viacnásobné inštrukcie, viacnásobné dáta)

Transputerový čip

Connection machine CM-5
(Thinking Machine Corporation)

Uzly - rýchle procesory + jednotka na spracovanie vektorov (129 miliónov oper/sek)

CM-5 (16 384 uzlov) 10^{12} oper/sek, priestor 600 m^2

CM-5 (3 point-to-point komunikačné siete)

- dátová sieť
- riadiaca sieť (globálna synchronizácia)
- diagnostická sieť

Algoritmické problémy

1. Implementácia systému na prepúšťanie správ (prenosový protokol, routovanie, vyhýbanie sa uviaznutiu)
2. Implementácia virtálne zdieľanej pamäte
3. Vyvažovanie záťaže (statické, dynamické)
4. Stabilita voči nedetekovateľným chybám (mechanizmus volieb)

1.4 Kooperujúce procesy

Návrh komplikovaných softwareových systémov môže byť často zjednodušený organizáciou softwaru ako množiny (sekvenčných) procesov.

Príklad : Conwayova konverzia záznamov

Problém pozostáva z načítania 80-znakových záznamov a zapísania tých istých dát ako 150-znakové záznamy

- po každom vstupe záznamu pridaj prázdny znak (medzeru)
- ** nahraď !
- každý výstupný záznam ukonči znakom EOR

Všetky funkcie musia byť vykonané v jednej slučke

Kapitola 2

Architektúra a jazyky

Software na implementáciu počítačových komunikačných sietí je zložitý. Často je štrukturovaný v acyklicky závislých moduloch nazývaných vrstvy.

2.1 Architektúra

Pod pojmom architektúra siete myslíme súbor vrstiev a príbuzných definícií všetkých rozhraní a protokolov.

Kompatibilita - zabezpečovaná štandardmi (ISO, IEEE)

Transmission control protocol (riadiaci protokol prenosu) / Internet protocol - TCP/IP - je množinou protokolov používaných v Internete. Je štruktúrovaný podľa vrstiev OSI modelu (môže byť použitý v WAN aj LAN). Najvyššie vrstvy obsahujú protokoly pre elektronickú poštu (simple mail-transfer protocol SMTP), prenos súborov (file-transfer protocol FTP), a obojsmernú komunikáciu pre vzdialené prihlasovanie (Telnet).

2.2 OSI referenčný model

ISO (International Standard Organization) ustanovila štandardy pre počítačové siete pre WAN. Štandard pre sieťové architektúry je OSI (Open-System Interconnection) referenčný model.

Algoritmické problémy

1. Atomickosť operácií s pamäťou
2. Problém producenta a konzumenta
3. Garbage collection (upratovanie pamäti)

Operačné systémy a programovacie jazyky poskytujú primitívy pre štruktúrovanú organizáciu medziprocesovej komunikácie

- Semafory

- Monitory
- Rúry (Pipes - UNIX)
- Posielanie správ (occam, Ada)

OSI referenčný model pozostáva zo siedmych vrstiev. Špecifikujú rozhrania medzi vrstvami a poskytujú štandardné distribuované algoritmy na implementáciu každej vrstvy.

1. *Fyzická vrstva* – vysiela postupnosti bitov cez komunikačný kanál
2. *Data-link vrstva* – na zakrytie nespoľahlivosti fyzickej vrstvy
3. *Sieťová vrstva* – komunikácia medzi každými pármi uzlov
4. *Transportná vrstva* – na zakrytie nespoľahlivosti sieťovej vrstvy
5. *Spojová vrstva* – poskytuje podporu pre uskutočnenie spojenia medzi procesmi v rôznych uzloch - t.j. prenos súborov
6. *Prezentačná vrstva* – vykonáva konverziu dát
7. *Aplikačná vrstva* – spĺňa konkrétne požiadavky užívateľa - prenos súborov, e-mail, nástenky, virtuálne terminály

2.2.1 OSI model v LAN : IEEE štandardy

Návrh OSI modelu je ovplyvnený architektúrou WAN.

Ak sieť spočíva v bežnej zbernici medzi všetkými uzlami, sieťová vrstva je takmer prázdna, transportná vrstva je veľmi zjednodušená.

IEEE štandardy

1. Fyzická vrstva
2. podvrstva riadenia stredného prístupu (Medium-acces-control)
3. podvrstva riadenia logických liniek (Logical-link-control)

2.3 Podpora jazykmi

Programovacie jazyky navrhnuté pre distribuované systémy

- krátky popis konštruktov

Paralelizmus : (komunikačná versus výpočtová zložitosť)

- staticky definované množiny procesov
- dynamická tvorba a ukončovanie procesov

Komunikácia

Primitívy posielania správ

- operácie vyšli, prijmi
- synchronizované posielanie správ
- point-to-point vysielanie alebo verejné alebo viacnásobné vysielanie

Zdieľaná pamäť

- komunikácia používa bežné premenné
- synchronizačné primitívy : semafory, monitory

Nedeterminizmus

- operácia prijmi je často nedeterministická
- strážený príkaz (na vyjadrenie nedeterminizmu)

2.4 Distribuované algoritmy

Distribuované a centralizované systémy sa líšia v dôležitých bodoch.

2.4.1 Distribuované versus centralizované algoritmy

1. Nedostatok vedomosti o globálnom stave (uzly v distribuovaných systémoch majú prístup len k ich vlastnému stavu a nie ku globálnemu stavu celého systému).

2. Nemožnosť globálneho riadenia času (Dočasný vzťah indukovaný na udalostiach ustavujúcich vykonanie distribuovaného algoritmu nie je úplný).

3. Nedeterminizmus (Vykonávanie distribuovaného systému je obvykle nedeterministické, kvôli možným rozdielom v rýchlosti vykonávania častí systému).

Príklad :

Spoľahlivá výmena informácii cez nespoľahlivé médium

- Dva procesy a,b , Procedúry riadenia siete (NCP) A,B, Informácia m,
- Nadviazanie a ukončenie komunikácie,
- Informačná jednotka sa môže stratiť, alebo duplikovať, NCP môže zlyhať (porucha) - reštartovanie v uzavretom stave

Spoľahivá výmena nie je dosiahnuteľná (Je jedno, ako zložito je NCP navrhnutá, nie je možné dosiahnuť úplne spoľahlivú komunikáciu)

- Inicializácia komunikácie procesom a
- NCP A a NCP B začnú konverzáciu, počas ktorej B doručí m do b
- NCP B sa zrúti a je reštartovaný v uzavretom stave

(V tejto situácii, ani A ani B nemôže povedať, či m bolo doručené, A nemôže pozorovať udalosti v B (nedostatok vedomosti o globálnom stave), B sa zrútil a bol reštartovaný v uzavretom stave.)

2.4.2 Konverzácia jednou správou

1. NCP A send <data,m>, notify, close
2. NCP B receive <data,m>, deliver m, close

2.4.3 Konverzácia dvomi správami

1. NCP A send <data, m>
2. NCP B receive <data, m>, deliver m, send <ack>, close
3. NCP A receive <ack>, notify, close

Zavedenie vypršania času, ak nie je prijatá žiadna správa

1. NCP A send <data, m>
2. NCP B receive <data, m>, deliver m, send <ack>, close
3. DN <ack> sa stratí
4. NCP A timeout, send <data, m>
5. NCP B receive <data, m>, deliver m, send <ack>, close
6. NCP A receive <ack>, notify, close

Možnosť duplicity bez strát

1. NCP A send <data, m_1>
2. NCP B receive <data, m_1>, deliver m_1, send <ack>, close
3. NCP A timeout, send <data, m_1>
4. NCP B receive <data, m_1>, deliver m_1, send <ack>, close
5. NCP A receive <ack>, notify, close
6. NCP A send <data, m_2>
7. DN <data, m_2> sa stratí
8. NCP A receive <ack> (step 2), notify, close

2.4.4 Konverzácia tromi správami

1. NCP A send <data, m>
2. NCP B receive <data, m>, deliver m, send <ack>
3. NCP A receive <ack>, notify, send <close>, close
4. NCP B receive <close>, close

- strata <ack> nevedie k duplikovaniu

- strata <close> môže spôsobiť opätovné vyslanie <ack>, môže prísť v nasledujúcej A konverzácii

1. NCP A send <data, m_1>
2. NCP B receive <data, m_1>, deliver m_1, send <ack>
3. NCP A receive <ack>, notify, send <close>, close
4. DN <close> sa stratí
5. NCP A send <data, m_2>
6. DN <data, m_2> sa stratí
7. NCP B retransmit <ack> (step 2)
8. NCP A receive <ack>, notify, send <close>, close
9. NCP B receive <close>, close

Jedna konverzácia zasahovala do inej konverzácie, toto môže byť vylúčené identifikačnými číslami konverzácií.

1. NCP A send <data, m, x>
2. NCP B receive <data, m, x>, deliver m, send <ack, x, y>
3. NCP A receive <ack, x, y>, notify, send <close, x, y>, close
4. NCP B receive <close, x, y>, close

B neoveruje platnosť <data, m, x> pred doručením m (krok 2) ⇒ duplicita

1. NCP A send <data, m, x>
2. NCP B receive <data, m ,x>, send <ack, x, y>
3. NCP A receive <ack, x, y>, notify, send <close, x, y>, close
4. DN <close, x, y> sa stratí
5. NCP B timeout, retransmit <ack, x, y>
6. NCP A receive <ack, x, y>, reply <nocom, x, y>
7. NCP B receive <nocon, x, y>, deliver m, close

B musí doručiť dáta aj keď A nepotrvdí spojenie s x,y ⇒ duplicita

1. NCP A send <data, m, x>
2. NCP A timeout, retransmit <data, m, x>
3. NCP B receive <data, m, x> (vyslaná v kroku 2), send <ack, x, y_1>
4. NCP A receive <ack, x, y_1>, notify, send <close, x, y_1>, close
5. NCP B receive <close , x, y_1>, deliver m, close
6. NCP B receive <data, m, x> (vyslaná v kroku 1), send <ack, x, y_2>
7. NCP A receive <ack, x, y_2>, reply ¡nocon, x, y_2>
8. NCP B receive <nocon, x, y_2>, in reply to <ack, x, y_2>, deliver m, close

2.4.5 Konverzácia štyrmi správami

Vzájomná dohoda na id. číslach konverzácie pred doručením dát.

1. NCP A send <data, m, x>
2. NCP B receive <data, m, x>, send <open, x, y>
3. NCP A receive <open x, y>, send <agree, x, y>
4. NCP B receive <agree, x, y>, deliver m, send <ack, x, y>, close
5. NCP A receive <ack, x, y>, notify, close

Úloha : Modifikuj protokol tak, aby A zaznamenalo a ukončilo po prijatí <noncon, x, y>, toto ochráni pred duplikáciou, avšak zavádza straty.

2.4.6 Konverzácia piatimi správami [Belsnes 1976]

Nestráca informácie a zavádza duplikácie len keď sa NCP skutočne zrúti.

2.5 Model distribuovaných výpočtov

2.5.1 Neformálny popis

Model Point-to-point

Distribuovaný systém - sieť G = (V, E), |V| = n procesorov, |E| = m kanálov

Procesor

- limitovaný lokálnou pamäťou (nezdieľaná)
- komunikácia so susedmi výmenou správ
- správanie môže byť popísané konečným počtom stavov a je riadené správami (Udalosť : príjem správy, lokálny výpočet - zmena stavu, vyslanie správy)

Formálny popis : Burns (1980), Lynch, Fischer (1981), Angluin (1980)

Predpoklady modelu :

- distribuovaný systém je asynchrónny
- asynchrónnosť
  - procesory
  - komunikácia
  - poradie správ
- distribuovaný systém nie je centrálne riadený (plne distribuovaný)
- každý procesor má jednoznačné identifikačné číslo (anonymné siete - bez i.č.)
- procesory komunikujú výmenou správ cez kanály

- čas potrebný na lokálny výpočet je zanedbateľný vzhľadom na čas potrebný na prenos správ.

Prídavné znalosti procesorov o topológii :

- striktne lokálna znalosť topológie (procesory vedia zoznam vlastných susedov)
- lokálna znalosť topológie + štruktúra siete (procesory vedia štruktúru siete : strom , kruh, ...)

Plne distribuovaný systém : predpokladajme že každý procesor má ten istý algoritmus

2.5.2 Formálny popis

Prechodový systém $S = (C, \rightarrow, I)$

- C množina konfigurácií
- $\rightarrow$ binárna relácia prechodu na C
- I počiatočné konfigurácie (podmnožina C)

Vykonanie S

maximálna postupnosť $E = (\gamma_0, \gamma_1, \ldots)$, kde $\gamma_0 \in I, \forall i \geq 0 : \gamma_i \rightarrow \gamma_{i+1}$

Terminálna konfigurácia γ

neexistuje taká δ že $\gamma \rightarrow \delta$

δ **je dosiahnuteľná z** γ (notácia $\gamma \rightsquigarrow \delta$)

$\exists \gamma = \gamma_0, \gamma_1, \ldots, \gamma_k = \delta s \gamma_i \rightarrow \forall 0 \leq i < k$

δ je dosiahnuteľná, ak je dosiahnuteľná z počiatočnej konfigurácie.

2.5.3 Systémy s asynchrónnym posielaním správ

Distribuovaný systém = množina procesov + komunikačný subsystém

Lokálny algoritmus procesu $(Z, I, \vdash^i, \vdash^s, \vdash^r)$

Z - množina stavov

I - počiatočné stavy (podmnožina Z)

$\vdash^i \subseteq Z \times Z, \vdash^s, \vdash^r \subseteq Z \times M \times Z$

Distribuovaný algoritmus pre množinu $\mathbf{P} = \{p_1, \ldots, p_N\}$ procesov je množina lokálnych algoritmov, jeden pre každý procesor v $\mathbf{P}$.

Prechodový systém indukovaný pod asynchrónnou komunikáciou

reprezentovaný distribuovaným algoritmom pre procesy $p_1, \ldots, p_N$ je $S = (C, \rightarrow, I)$ kde

1. $C = \{(c_{p_1}, \ldots, c_{p_N}; M) | (\forall p \in \mathbf{P} : c_p \in Z_p) \wedge M \in \mathbf{M}(M)\}$
2. $\rightarrow = (\bigcup_{p \in \mathbf{P}} \rightarrow_p)$, kde $\rightarrow_{p_i}$ je množina párov $(c_{p_1}, \ldots, c_{p_i}, \ldots, c_{p_N}, M_1)$, $(c_{p_1}, \ldots, c'_{p_i}, \ldots, c_{p_N}, M_2)$ pre ktoré
   - $(c_{p_i}, c'_{p_i}) \in \vdash^i_{p_i}$ a $M_1 = M_2$,
   - pre nejaké $m \in M$, $(c_{p_i}, m, c'_{p_i}) \in \vdash^s_{p_i}$ a $M_2 = M_1 \cup m$,
   - pre nejaké $m \in M$, $(c_{p_i}, m, c'_{p_i}) \in \vdash^r_{p_i}$ a $M_1 = M_2 \cup m$,
3. $I = \{(c_{p_1}, \ldots, c_{p_N}; M) | (\forall p \in \mathbf{P} : c_p \in I_p) \wedge M = 0\}$.

2.5.4 Systémy so synchrónnym posielaním správ

Prechodový systém indukovaný pod synchrónnou komunikáciou

reprezentovaný distribuovaným algoritmom pre procesory $p_1, \ldots, p_N$ je $S = (C, \rightarrow, I)$, kde

1. $C = \{(c_{p_1}, \ldots, c_{p_N}) | (\forall p \in \mathbf{P} : c_p \in Z_p)\}$
2. $\rightarrow = (\bigcup_{p \in \mathbf{P}} \rightarrow_p) \cup (\bigcup_{p,q \in \mathbf{P}: p \neq q} \rightarrow_p q)$, kde
   - $\rightarrow_{p_i}$ je množina párov $(c_{p_1}, \ldots, c_{p_i}, \ldots, c_{p_N})$, $(c_{p_1}, \ldots, c'_{p_i}, \ldots, c_{p_N})$ pre ktoré $(c_{p_i}, c'_{p_i}) \in \vdash^i_{p_i}$
   - $\rightarrow_{p_i p_j}$ je množina párov $(\ldots, c_{p_i}, \ldots, c_{p_j}, \ldots)$, $(\ldots, c'_{p_i}, \ldots, c'_{p_j}, \ldots)$ pre ktoré je správa $m \in M$ taká, že $(c_{p_i}, m, c'_{p_i}) \in \vdash^s_{p_i} a (c_{p_j}, m, c'_{p_j}) \in \vdash^s_{p_j}$
3. $J = \{(c_{p_1}, \ldots, c_{p_n}) | (\forall p \in \mathbf{P} : c_p \in I_p)\}$.

2.5.5 Overovanie vlastností prechodových systémov

Vlastnosť bezpečnosti "Tvrdenie P je pravdivé v každej konfigurácii každého vykonania algoritmu" (P je vždy pravdivé)

Tvrdenie P je invariant prechodového systému $S = (C, ->, I)$, ak

1. pre všetky $\gamma \in I : P(\gamma)$, a
2. $\{P\} \rightarrow \{P\}$

Veta : Ak P je invariant S, potom P platí pre každú konfiguráciu každého vykonania S.

Dôkaz : Nech $E = (\gamma_0, \gamma_1, \ldots)$ je vykonanie S. Indukciou ukážeme, že $P(\gamma_i)$ platí pre každé i.
Po prvé , $P(\gamma_0)$ platí, pretože $\gamma_0 \in I$.
Po druhé, predpokladajme že $P(\gamma_i)$ platí a $\gamma_i \rightarrow \gamma_{i+1}$ je v E. Potom $P(\gamma_{i+1})$. ♠

Veta : Nech Q je invariant S a predpokladajme $Q \Rightarrow P$ (pre každé $\gamma \in C$). Potom P platí v každej konfigurácii každého vykonania S.

Dôkaz : Nech S je prechodový systém a P, Q sú tvrdenia. P je Q-derivát ak
1) pre všetky $\gamma \in J : Q(\gamma) \Rightarrow P(\gamma)$ 2) $\{Q \wedge P\} \rightarrow \{Q \Rightarrow P\}$

Veta : Ak Q je invariant a P je Q-derivát, potom $Q \wedge P$ je invariant.

Vlastnosť životnosti: Tvrdenie P je pravdivé v niektorých konfiguráciách každého vykonania algoritmu. (P môže byť pravdivé)

- S - prechodový systém P - predikát
- **term** - predikát - pravdivý vo všetkých terminálnych konfiguráciách
  - nepravdivý vo všetkých neterminálnych konfiguráciách
- **uviaznutie** (deadlock) - vykonávanie dosiahne terminálnu konfiguráciu a "riešenie" P nebolo dosiahnuté
- **správna terminácia** - terminácia je povolená, ak sa dosiahol cieľ

Systém S správne terminuje (alebo neobsahuje uviaznutie) ak predikát (term $\Rightarrow$ P) je vždy pravdivý v S.

Dobre založená množina je množina s usporiadaním $<$, pre ktoré neexistujú nekonečné klesajúce postupnosti.
Čiastočné usporiadanie $(W, <)$ je dobre založené ak neexistuje nekonečná klesajúca postupnosť $w_1 > w_2 > w_3 > \ldots$

Príklad : Dobre založené množiny :
prirodzené čísla s bežným usporiadaním
n-pások prirodzených čísel s lexikografickým usporiadaním

Nech je daný prechodový systém S a tvrdenie P. Funkcia f z C do dobre založenej množiny W sa nazýva normálna funkcia (s ohľadom na P) ak pre každý prechod $\gamma \rightarrow \lambda$, $f(\gamma) > f(\lambda)$ alebo $P(\lambda)$.

Veta : Ak S správne terminuje a existuje normálna funkcia f (w.r.t. P), potom P je pravdivé v nejakej konfigurácii nejakého vykonania S.

Dôkaz : Nech E=$(\gamma_0, \gamma_1, \ldots)$ je vykonávanie S.
Ak E je konečná, jej posledná konfigurácia je terminálna a ako term $\Rightarrow$ P je vždy pravdivá v S, P platí v tejto konfigurácii.
Ak E je nekonečná, nech E' je najdlhší prefix z E ktorý neobsahuje konfiguráciu v ktorej je P pravdivé, a nech s je $(f(\gamma_0), f(\gamma_1), \ldots)$ pre všetky konfigurácie γ_i ktoré sa objavia v E'.
s je klesajúca postupnosť, a preto s je konečná. Toto implikuje že E' je konečný prefix $(\gamma_0, \gamma_1, \ldots, \gamma_k)$ E, podľa výberu E', $P(\gamma_{k+1})$ platí.♠

Veta : Ak S správne terminuje a existuje číslo K také, že každé vykonanie obsahuje najviac K prechodov, potom P je pravdivé v nejakej konfigurácii každého vykonania.

2.6 Zložitosť

Topológia siete Sieť = množina procesov + komunikačný subsystém (reprezentovaný ako graf)

1. Kruhy
2. Stromy
3. Hviezdy
4. Kliky (kompletné grafy)
5. Hyperkocky
   $HC_n = (V, E)$, N = 2_n uzlov, V - množina bitových reťazcov dĺžky n, existuje hrana medzi uzlami b, c, akk bitový reťazec b, c sa líši práve v jednom bite.

* Statická alebo dynamická topológia

Vlastnosti kanálov

1. Spoľahlivosť
2. FIFO vlastnosť
3. Kapacita kanálu

Vedomosti procesu

1. Topologická informácia

   číslo procesorov, polomer, topológia

   zmysel pre smer = ak v grafe poznáme konzistentné označenie hrán so smermi
2. Identita procesu (pomenovaná sieť, anonymná sieť)
3. Susedné identity (priame versus nepriame adresovanie)

Zložitosť distribuovaných algoritmov

1. Zložitosť vzhľadom na počet správ
2. Bitová zložitosť
3. Časová zložitosť

   idealizované meranie času udalostí výpočtu:

   (a) čas pre spracovanie udalosti je nulový.

   (b) prenosový čas(t.j. čas medzi vyslaním a prijatím správy) je najviac jedna časová jednotka.

   Časová zložitosť algoritmu je čas spotrebovaný výpočtom vzhľadom na tieto predpoklady.
4. Priestorová zložitosť

2.7 Úlohy

Úloha 2.1 Definujte model distribuovaného systému v ktorom môžu byť správy posielané synchrónne aj asynchrónne zároveň.

Úloha 2.2 Udajte prechodový systém S a tvrdenie P také že P je vždy pravdivé v S, ale nie je invariant S.

Úloha 2.3 Predpokladajme, že P_1 a P_2 sú invarianty systému S. Dokážte, že $(P_1 \vee P_2)$ a $(P_1 \wedge P_2)$ sú invarianty.

Predpokladajme, že P_1 a P_2 sú Q-deriváty systému S. Dokážte že $(P_1 \vee P_2)$ a $(P_1 \wedge P_2)$ sú Q-deriváty systému S.

Kapitola 3

Distribuovaný výber

Problém výberu (nazývaný aj voľba šéfa) v distribuovaných sieťach je nasledovný : začať v konfigurácii, kde všetky procesory sú v tom istom stave, a dospieť do konfigurácie, kde práve jeden procesor je v stave "leader" a všetky ostatné procesory sú v stave "lost".

- Súčasť rôznych algoritmov (minimálna kostra, prechádzanie grafu)
- Inicializačná procedúra po zrútení systému

Symetria
Procesory nemajú identifikačné čísla.

- Anglain(1980) - Neexistuje deterministický algoritmu.
- Frederickson, Santoro (86) - symetria môže byť porazená s pravdepodobnosťou 1 v priemernom čase time O(n) a komunikácii O(n) bitov.

Výber bol študovaný s ohľadom na nasledujúce kritériá:

- topológia siete (kruh, kompletná sieť, ľubovoľná sieť,...)
- synchrónna versus asynchrónna siet
- informácie o topológii v procesoroch
  (zmysel pre globálnu orientáciu, veľkosť siete, ...)

Jednosmerné kruhy			
	Dolné ohraničenie	Horné ohraničenie	
		Priemerný prípad	Najhorší prípad
Le Lann (1979)		n^2	n^2
Chang, Roberts (1979)		$0.69n \log n^*$	$0.5n^2$
Peterson (1982)		$0.943n \log n$	$1.44n \log n$
Dolev, Klawe, Rodeh (1982)		$0.967n \log n$	$1.356n \log n$
Pachl, Korach, Rotem (1982)	$0.69n \log n$		
Higham, Przytzcka (1993)			$1.271n \log n$

Dvojsmerné kruhy			
	Dolné ohraničenie	Horné ohraničenie	
		Priemerný prípad	Najhorší prípad
Gallager, Humblet, Spira (1979)			$5n \log n$
Hirschberg, Sinclair (1980)			$8n \log n$
Burns (1980)	$0.25n \log n$		$3n \log n$
Franklin (1982)			$2n \log n$
Santoro, Korach, Rotem (1982)			$1.89n \log n$
Pachl, Korach (1982)	$0.125n \log n$		
Bodlaender, van Leeuwen (1986)		$0.488n \log n^*$	$0.25n^2$
Santoro, Korach, Rotem (1981)		$0.517n \log n$	$0.5n^2$
van Leeuwen, Tan (1985)			$1.44n \log n$
Moran, Shalom, Zaks (1985)			$1.44n \log n$

$*$ Vedomosti o globálnej orientácii

Idea 1 : Založená na hladaní maxima.
Identifikačné čísla obiehajú niektorým smerom v kruhu a zastavia v procesore, ktorý vie, že nemôže byť maximálny.

Chang , Roberts (1979) {Jednosmerné kruhy, priemerný prípad}
Identifikačné čísla všetkých procesorov sú vyslané tým istým smerom a zastavia v prvom "väčšom" procesore.

Veta : Vzdialenosť k prvému väčšiemu záznamu v náhodnej postupnosti dĺžky n je $H_n - 1 \approx 0.69 \log_2 n$.

Idea 2 : Založená na etapách lokálnych výberov.
Počas jednej etape vyšlú všetky aktívne procesory ich identifikačné čísla najbližším procesorom vľavo a vpravo. (Neaktívne procesory jednoducho vysielajú správy ďalej). Keď aktívny procesor zistí, že má väčšieho aktívneho "suseda" vľavo alebo vpravo, stane sa neaktívnym.

* Fanklin (82) : # etáp $\leq \log_2 n$; jedna etapa - výmena 2n správ

* Maximum v kruhu aktívnych a neaktívnych procesorov - aktívny procesor ktorý je väčší ako aktívne "susedné" procesory.

Veta : Priemerný počet maxím v permutácii n prvkov je $\frac{1}{3}(2n - 1)$.

3.1 Dvojsmerné kruhy, priemerný prípad.

3.1.1 Algoritmus - P

H.L. Bodlaender, J. van Leeuwen (85) algoritmus $0.52n \log_2 n$
{Každý procesor p si pamätá najväčšie identifikačné číslo ktoré videl v MAXID(p)}

Inicializácia

MAXID(p) := identifikačné číslo procesora p; {id.č. jednoznačne označuje procesor}
vyber smer DIR $\in$ {vpravo, vľavo} s pravdepodobnosťou $\frac{1}{2}$, vyšli správu <p> v smere DIR po kruhu,

Výber {Opakuj nasledujúce kroky, pokiaľ nie je koniec výberu signalizovaný prijatím správy <elected>}
ak sú prijaté dve správy zprava a zľava súčasne,
potom ignoruj menšiu správu a postupuj, ako keby bola prijatá len väčšia správa
ak správa <q> je prijatá od suseda, potom
if q > MAXID(p)
then MAXID(p) := q; posielaj správy <q> ďaľej
else if q=MAXID(p) then {p vyhralo výber} pošli <elected> po kruhu

Inaugurácia Ak je prijatá správa <elected>, výber sa skončil a v MAXID(p) je identifikačné číslo šéfa. Stop.

Veta : Algoritmus-P používa

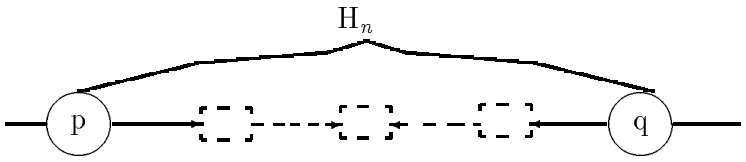
(i) $\sim \frac{1}{2}n^2$ správ v najhoršom prípade

(ii) (najviac) $\sim \frac{3}{4}n.H_n \approx 0.52n \log_2 n$ správ v očakávanom prípade

Dôkaz:

(i) Najhorší prípad nastane v kruhu s identifikačnými číslami $n, n-1, \ldots, 1$ v smere doprava

(ii)

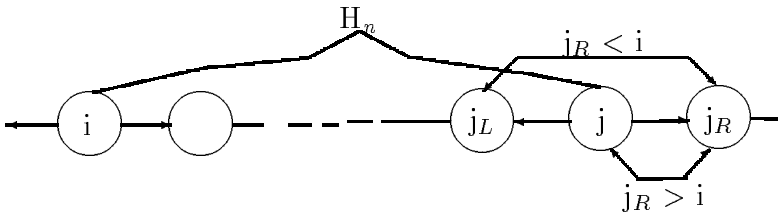


- správa vygenerovaná procesorom p bude zlikvidovaná prvým väčším procesorom q
- Ak q poslal svoje id.č. ku p, potom sa správy stretnú "na polceste" a p-správa je zrušená. Je pravdepodobnosť $\frac{1}{2}$ že p-správa potrebuje prejsť len polovičnú vzdialenosť. Očakávaný počet p-správ bude $\frac{1}{2}H_n + \frac{1}{2}.\frac{1}{2}H_n = \frac{3}{4}H_n$
- Ak q vyšle svoje id.č. smerom od p, je možné že druhý väčší procesor sa rozhodne poslať svoje id.č. ku p. Ak toto nastane, zruší q-správu a je možné, že zastaví p-správu pred dosiahnutím procesora q. Preto očakávaný počet správ bude o trohu menej ako $\frac{3}{4}H_n$ na procesor.
- Úplný počet vymenených správ je menej než $\frac{3}{4}nH_n + n \sim \frac{3}{4}nH_n \sim 0.52n \log_2 n$.

3.1.2 Algoritmus-D

{Podobný ako Algoritmus-P, okrem toho, že pre každý procesor p je etapa Inicializácia nahradená nasledujúcou etapou}

Inicializácia vyšli správu <p> obom susedom na kruhu,
čakaj na správy <p_1>,<p_2> od oboch susedov,
MAXID(p) := max$\{p_1, p, p_2\}$;
if MAXID(p) = p then
if $p_1 < p_2$
then vyšli správy <p> doľava
else vyšli správy <p> doprava



- $j_L < i$
- ak $j_R < i$: vymeň (j_L, j_R)
- ak $j_R > i$: vymeň (j, j_R)

3.2 Dvojsmerné kruhy, najhorší prípad

J. van Leeuwen, R.B.Tan (1985) algoritmus $1.44n \log n$ + O(n)

Črty algoritmu :

- predpoklad "half-duplex" (kanály sú obojsmerné ale môžu prenášať správu len jedným smerom naraz)
- iniciácia hladania
- logické radenie hľadaní - fázy
- 2 typy "nezvyčajných" situácií
  - kolízia dvoch správ
  - náraz (správa - procesor)

3.2.1 Algoritmus výberu

{Algoritmus popisuje akcie ľubovoľného procesora na dvojsmernom kruhu s half-duplexnými linkami ako sú vyžadované pre výber šéfa}

1.Inicializácia

{procesor q má 3 premenné MAXID, DIR, PNUM}
vyšli správu <MAXID(q), 0> obom susedom;
PNUM := 0;
čakaj na zodpovedajúce správy $<u_1$,0>, $<u_2$,0> od oboch susedov;
if u_1> MAXID(q) & u_2 > MAXID(q)
then MAXID(q):= $\max(u_1, u_2)$;
DIR := smer $\min(u_1, u_2)$;
goto active {q bude v stave aktívny}
else goto observant {q bude v stave pozorovateľ}

2.Výber

A. {Procesor q je aktívny}
PNUM := p + 1; {číslo fázy p je buď uložené v PNUM alebo v dočasnom registri}
vyšli správu <MAXID(q), PNUM> smerom DIR;
goto observant; {q bude v stave pozorovateľ}

B. {Procesor q je pozorovateľ}
prijmi správu (s) z jedného alebo oboch smerov;
zruš správu <v,p> prijatú s p < PNUM;
zruš správu ktorá nemá najvyššiu p-hodnotu spomedzi správ;
case # zvyšných správ

0 : goto observant;
1 : {nech prijatá správa je <v,p>, kde nutne p ≥ PNUM}
if p = PNUM(q)
then case "test ako indikované"
v = MAXID(q): goto inaugurate; v < MAXID(q):
DIR:=smer z ktorého bola správa prijatá
goto active; {Náraz}
v > MAXID(q):
{zastav správu, priradenie MAXID:=v je voliteľné}
goto observant
else {p > PNUM a správa musí byť prenesená ďalej}
PNUM := p;
MAXID(q) := v;
DIR := smer, ktorým správa smerovala;
vyšli správu <MAXID(q), p> do smeru DIR;
goto observant

2 :{Nech dve prijaté správy sú <v_1, p>, <v_2, p>, nutne z opačných smerov a s p $\geq$ PNUM}
case "test ako indikované"
$v_1 = v_2$: PNUM := p; goto inaugurate;
$v_1 \neq v_2$: {kolízia}
MAXID(q) := max(v_1, v_2);
DIR := smer min(v_1, v_2);
goto active

3. Inaugurácia {Procesor q príde do tejto finálnej fázy keď algoritmus skončí, a MAXID(q) obsahuje identitu (jediného) šéfa.}

3.2.2 Analýza algoritmu

Lema 1 : Ak procesor u prijme správu $m \equiv$ <v, p>, p = PNUM(u), potom m musela prijsť zo smeru (t.j. linky) opačnej k tej, ktorou u naposledy vyslal správu.

- A_p je množina procesorov u ktoré sú aktívne vo fáze p
- $A_p^{\sim}$ je množina farieb procesorov u ktoré sú aktívne vo fáze p ("aktívne farby")

Lema 2 : Pre všetky $p \geq 1 : A_p^{\sim} \supseteq A_{p+1}^{\sim}$

Lema 3 : {c - kolízia ; d - nie je kolízia}

(a) Medzi každými dvomi c-aktívnymi procesormi vo fáze p existujú aspoň dva neaktívne procesory v tej istej fáze p.

(b) c-aktívny procesor vo fáze p nemôže byť c-aktívny v nasledujúcej fáze p+1.

(c) d-aktívny procesor vo fáze p nemôže byť c-aktívny vo fáze p+1.

(d) Ak dva priľahlé procesory sú aktívne v tej istej fáze p, potom si nevysielajú správy navzájom priamo v tejto fáze p.

Dôkazy : (a) Podľa Lemy 1 u' nemôže prijať správu od u v p-1, teda u' sa nemôže stať c-aktívnym vo fáze p. u" nemôže vyslať správu ku u' vo fáze p-1, preto u' sa nemôže stať c-aktívnym vo fáze p.

(b),(c) vyplývajú priamo z Lemy 1.

(d) Uvažujme p = 1 : Aktívny procesor musí byť "lokálne minimum" $\Rightarrow$ neexistujú dva susedné aktívne procesory.
Uvažujme p > 1 : Z Lemy 3(a), u, v nemôžu byť c-aktívne.
obrázok
Preto v nevyšle správu vo fáze p ku u! ($\gamma <$ MAXID(v))
obrázok
Potom u,v vyslal < α ,p-1>, <β, p-1>, kde $\alpha < \beta$ (vo v), $\alpha > \beta$ (v u), spor!

Lema 4 : Pre všetky p $\geq$: ak $A_p \neq \emptyset$ a šéf sa nenašiel vo fáze p, potom $A_{p+1} \neq \emptyset$.

Dôkaz sporom. Nech p je najväčšie číslo fázy pre ktorú $A_p \neq \emptyset$ (teda p$\geq$1) ale $A_{p+1} = \emptyset$, pretože šéf nebol nájdený vo fáze p.

- $u \in A_p$:
  - ak u je jediný aktívny na kruhu; u je šéf $\Rightarrow$ spor.
- $u_1, \ldots, u_k \in A_p (k \geq 2)$:
  - ak u_i, u_{i+1} vyslali správy <d_i,p>,<d_{i+1},p> sebe navzájom $\Rightarrow$ kolízia v niektorom stredovom u' $\Rightarrow$ u' $\in A_{p+1} \Rightarrow$ spor.
  - ak u_i, u_{i+1} vyslali <α_i,p>,<α_{i+1},p> tým istým smerom, $\alpha_i < \alpha_{i+1} \Rightarrow$ zrážka v $u_{i+1} \Rightarrow u_{i+1} \in A_{i+1} \Rightarrow$ spor.
  - $\alpha_i \geq \alpha_{i+1}$ pre všetky i, ale $\alpha_{k+1} \equiv \alpha_1 \Rightarrow \alpha_1 = \alpha_2 = \ldots = \alpha_k \Rightarrow$ správa <α_i,p> prišla do $u_{i+1} \Rightarrow u_{i+1}$ goto inauguration $\Rightarrow u_{i+1}$ je šéf $\Rightarrow$ spor.

Lema 5 : Predpokladajme, že dva aktívne procesory u, u' $\in A_p$ majú rovnakú farbu α. Potom

(a) u,u' musia byť priľahlé ,teda u,u' sú oddelené len neaktívnymi procesormi,

(b) všetky procesory medzi u, u' (v jednom smere) majú rovnakú hodnotu α v ich registri MAXID,

(c) u, u' vyšlú svoje fáza-p správy v opačných smeroch, t.j. "smerom od hrán α-zafarbeného intervalu".

Lema 6 : Ak procesor príde do inauguračného stavu vo fáze p, potom:

(a) je jedný taký,

(b) $|A_p^{\sim}| = 1$,

(c) všetky procesory na kruhu majú rovnakú hodnotu v ich MAXID registri.

Prípad I : u má hodnotu α v MAXID a prijme <α,p> $\{\alpha \in A_p^{\sim}\}$

- Predpokladajme, že existuje jediný α-zafarbený aktívny procesor vo fáze p $\Rightarrow$ neexistuje aktívny procesor s $\alpha' \neq \alpha \Rightarrow |A_p^{\sim}| = 1$.
- Predpokladajme, že existujú α-zafarbené aktívne procesory v, v' $\in A_p$ (Podľa Lemy 5 sú najviac dva!) Lema 5 $\Rightarrow$ u má farbu α, ale <α, p> nemôže dosiahnť u $\Rightarrow$ tento podprípad nemôže nastať!

Prípad II : PNUM(u) $\leq$ p, u prijme správy <α,p> z oboch strán {kolízia}

Lema 7 : Existuje 1-1 mapovanie z aktívnych farieb vo fáze p do aktívnych farieb vo fáze p-2 ktoré už neboli aktívne vo fáze p-1.

Lema 8 : Algoritmus vždy končí po konečnom počte fáz a (preto) je korektným distribuovaným algoritmom výberu.

- Lema 2 + Lema 7 $\Rightarrow \forall p \geq 2 : |A_p^{\sim}| \leq |A_{p-2}^{\sim}| - |A_{p-1}^{\sim}|$
- $|A_0^{\sim}| = n, |A_1^{\sim}| \leq \frac{n}{2}$. Teda $|A_p^{\sim}| < |A_{p-2}^{\sim}|$ ak nepoznáme šéfa vo fáze p-1, teda $|A_{p-1}^{\sim}| > 0$.

Lema 9 : Algoritmus skončí počas $1.44 \log n$ fáz a používa najviac $1.44n \log n + O(n)$ správ.

- Predpokladajme, že algoritmus končí po t fázach. Z Lemy 6 $\Rightarrow |A_{t-1}^{\sim}| = 1$.
- Lema 2 + Lema 7 $\Rightarrow \forall p \geq 2 : |A_p^{\sim}| \leq |A_{p-2}^{\sim}| - |A_{p-1}^{\sim}|$
  Teda $|A_{p-2}^{\sim}| \geq |A_{p-1}^{\sim}| + |A_p^{\sim}|$
- Pre $0 \leq i \leq t$ platí $F_i \leq |A_{t-i}^{\sim}|$.
  - pre i = 0,1 platí $F_0 \leq |A_t^{\sim}|, F_1 \leq |A_{t-1}^{\sim}|$
  - predpoklad pre fázy $\leq$ i
  - uvažujme i+1 $\leq$ t potom
    $|A_{t-(i+1)}^{\sim}| \geq |A_{t-i}^{\sim}| + |A_{t-(i-1)}^{\sim}| \geq F_i + F_{i-1} = F_{i+1}$
- Z toho vyplýva, že $F_t \leq |A_0^{\sim}| \leq n$
- Zo známeho výpočtu F_t :

$$F_t = \left(\frac{1+\sqrt{5}}{2} \right)^t / \sqrt{5} + O(1)$$

usudzujeme $\{\frac{1+\sqrt{5}}{2} = 1.61803\ldots\}$

$$t \leq \frac{\log_2 n}{log_2(\frac{1+\sqrt{5}}{2})} + O(1) \approx 1.44 \log_2 n$$

- Každá fáza ($\neq$ inicializácia) potrebuje najviac n správ.
- Inicializácia potrebuje 2n správ.

3.2.3 Modifikácia algoritmu pre všeobecné dvojsmerné kruhy

$$\alpha > \beta$$

- u - aktívny, v - neaktívny

Modifikácia : Zavedenie 1-bitového poľa navyše do správy

(i) u prijme <β,p,0> a zruší ju {w aktívne v p}

(ii) v prijme <α,p,1>. Procesor v sa stane aktívny vo fáze p+1 a vyšle <...,p+1,1> požadovaným smerom. {Platí to aj pre $\alpha < \beta$ alebo pre u neaktívne v p, v aktívne v p}

- u,v - neaktívne vo fáze p:
  <α,p>, <β,p>: "kolízia" je simulovaná ako d-aktivácia v u ($\alpha > \beta$) alebo vo v ($\alpha < \beta$)

- u,v - inaugurácia ($\alpha = \beta$) :
  u,v majú rovnakú hodnotu MAXID, teda vyberú ten istý procesor ako šéfa

Lema 10 : Modifikovaná verzia algoritmu Výber je korektný distribuovaný algoritmus výberu pre plne asynchrónne dvojsmerné kruhy.

Lema 11 : Modifikovaná verzia algoritmu Výber sa ukončí počas $1.44 \log n$ fáz a použije najviac $1.44n \log n + O(n)$ správ.

3.3 Jednosmerné kruhy, Najhorší prípad

D.Doley, M.Klawe, M.Rodeh (1982) algoritmus $1.356n \log n + O(n)$
Algoritmus pre aktívny procesor p:

1. Procesor p vyšle správu <1,MAXID(p)>

2. Ak správa <1,i> dorazí do procesora p :

   if $i \neq MAXID(p)$
   then p vyšle správu <2,i> a LEFT(p):= i
   else halt {MAXID(p) je globálne maximum}

3. Ak správa <2,j> dorazí do p:

   if LEFT(p) > j $\wedge$ LEFT(P) > MAXID(p)
   then MAXID(p):= LEFT(p) a vyšle správu <1,MAXID(p)>
   else procesor p sa stane pasívny

$$\sim 2n \log n + n$$

3.3.1 Úpravy algoritmu

1. Ak MAXID(u) > MAXID(v)
   potom procesor v nevyšle správu typu 2

2. Ak existuje procesor x medzi u,v taký, že MAXID(x) > MAXID(U),
   potom x nevyšle správu typu 2 z u ku v

3. Zabezpečiť, že správy typu 2 inicializované vo fáze p nebudú nikdy cestovať ďalej, ako 2^p od ich iniciátora
   Pridáme počítadlo s počiatočnou hodnotou 2^p ku každej správe typu 1 inicializovanej vo fáze p. Pasívny procesor zníži hodnotu počítadla o 1. Ak pasívny procesor q prijme správu typu 1 s hodnotou počítadla 1 a id.č. $\leq$ MAXID(q), potom

- vyšle správu typu 1
- prepne sa do čakacieho stavu
- MAXID(q) := identifikačné číslo k procesoru

Ak q prijme ako nasledujúcu správu typu 2, tak sa prepne do aktívneho stavu, inak zostane pasívny

3.3.2 Algoritmus

{Každý procesor v : PNUM - počiatočne 0; STATE ∈ {active, waiting, passive}, ID, MAXID - počiatočne ID} {Algoritmus používa dva typy správ tvaru <1,i,PNUM,počítadlo>, <2,j>, kde $0 \leq$ počítadlo $\leq 2^{PNUM}$}
Správanie sa procesora:

1. Aktívny procesor v inicializuje správu <1,MAXID(v),PNUM(v),$2^{PNUM(v)}$>.
   Ak dorazí správa tvaru <1,i,p,c>, potom :
   if i>MAXID(v)
   then MAXID(v) := i; STATE(v) := "waiting"
   else STATE(v) := "passive"; send <2,MAXID(v)>
   {Všimnite si, že správy typu 2 v predošlom algoritme boli <2,v>, teraz <2,MAXID(v)>. Toto je dôležité v modifikácii (2). Pasívny procesor x prijímajúci <2,MAXID(v)> vie, že ju nepotrebuje vyslať, ak MAXID(v) < MAXID(x). Toto je kvôli tomu, že procesor čakajúci na tú správu typu 2 aby sa stal aktívnym v nasledujúcej fáze môže byť reprezentovaný MAXID(v) ak sa stane aktívny, a MAXID(v) nie je očividne globálne maximum.}
   {Všimnite si, že aktívny procesor nikdy neprijme správy typu <2,j>. Také správy sú prijímané len čakajúcimi alebo neaktívnymi procesormi.}

2. Čakajúci procesor môže prijať správu oboch typov.

   a. {Správa typu 2 dorazí, <2,j> s j=MAXID(v)}
      PNUM(v):=PNUM(v) + 1; STATE(v):= "active"; goto step 1

   b. {Správa typu 1 dorazí}
      STATE(v):= "passive"; goto step 3a

3. Pasívny procesor v

   a. {ak dorazí <1,i,p,c>}
      if (i≥ MAXID(v), c≥1)
      then MAXID(v):= i;
           if c>1
           then send <1,i,p,c-1>
           else STATE(v) := "waiting";
                send <1,i,p,0>;
                PNUM(v):=p
      else send <1,i,p,0>

b. {ak dorazí <2,j>}
if j ≥ MAXID(v)
then send <2,j>

$$\sim 1.5n \log_2 n + O(n)$$

3.3.3 Korektnosť a zložitosť

F1 : Počas každej fázy každý procesor vyšle práve dve správy.

F2 : Ak u,v sú rôzne procesory ktoré sú v tej istej fáze p, potom MAXID(u)≠ MAXID(v).

F3 : Ak u je aktívny vo fáze p a MAXID(u) je lokálne maximum aktívnych procesorov vo fáze p, potom procesor v, ktorý je najbližšie ku u na kruhu z týchto aktívnych procesorov, bude aktívny vo fáze p+1 a bude mať MAXID(v) vo fáze p+1 rovné MAXID(u) vo fáze p.

F4 : Ak existujú aspoň dva aktívne procesory vo fáze p, tak potom pre každý taký aktívny procesor v, správa <1,i> ktorú prijme vo fáze p má i ≠ MAXID(v) vo fáze p.

Lema 1 : Počet aktívnych procesorov vo fáze p+1 je najviac polovica počtu aktívnych procesorov vo fáze p

- počet lokálnych maxím $\leq \frac{1}{2}n$
- F2+F3 zabezpečujú, že počet aktívnych procesorov vo fáze p+1 je práve počet aktívnych procesorov vo fáze p ktorých hodnoty sú lokálne maximá

Lema 2 : Počte správ vyslaných algoritmom je najviac $2n \log n + n$.

- počet fáz : $log_2 n + 1$
- počet správ v jednej fáze : $2n$
- finálna fáza : n správ

A(p): množina procesorov ktoré sú aktívne na začiatku fázy p

p-max(v): hodnota MAXID(v) na začiatku fázy p

Lema 1 : Ak u,v ∈ A(p), u ≠ v, potom p-max(v).

- p=0: pravdivé tvrdenie
- predpokladajme, že tvrdenie platí pre p-1
- procesory v A(p-1) majú rôzne hodnoty MAXID ⇒ správy typu 2 vo fáze p-1 majú rôzne hodnoty ⇒ procesory z A(p), pridané správami typu 2, budú mať hodnoty p-max rovné správe typu 2 ⇒ procesory v A(p) majú rôzne hodnoty p-max.

Lema 2 : Počas fázy p každá správa typu 2 prejde vzdialenosť najviac 2^p od procesora ktorý ju vytvoril.

- Nech u,v $\in$ A(p), u,v sú susedia
  - ak vzdialenosť (u,v) $> 2^p$: správa typu 1 prejde vzdialenosť 2^p, prepnutie do čakajúceho stavu, správa typu 2 zastane.
  - ak vzdialenosť (u,v) $\leq 2^p$:
    - ak MAXID(v) < MAXID(u) $\Rightarrow$ v čakajúci procesor, zastaví správu typu 2
    - ak MAXID(v) > MAXID(u) $\Rightarrow$ hodnota správy typu 2 < MAXID(v) $\Rightarrow$ procesor v nemôže vyslať správu typu 2

Lema 3 : Počas ľubovoľej fázy prejde po hrane najviac jedna spáva každého typu.

Lema 4 : Nech M je najväčšia z hodnôt ID(v) všetkých procesorov v. Potom ak nejaké u $\in$ A(p) má p-max(u) = M, tak existuje procesor v $\in$ A(p+1) s (p-1)-max = M.

- indukciou vzhľadom na p

Lema 5 : Vždy existuje dajaký procesor v, v $\in$ A(p) s p-max=M.

d(u,v) : vzdialenosť z u od v na kruhu (v smere hodinových ručičiek)

u < v < w : znamená, že prejdeme po kruhu z u, cez v do w

origin(j) : procesor v s j ako svojou pôvodnou hodnotou (t.j. IČ(v) =j)

Lema 6 : Nech v je procesor v A(p).

(a) d(origin(p-max(v), v) = $2^p - 1$

(b) p-max(y) = p - max(v) pre každý procesor y s origin(p-max(v)) $\leq$ y $\leq$ v

(c) Ak w je rôzny procesor od v z A(p), potom d(w,v,) $\geq 2^p$.

Korektnosť 7 : Pre každé p, $|A(p)| \leq n/2^p$.

- Priamy dôsledok Lemy 6(c).

Lema 8 : Nech u je procesor ktorý iniciuje správu typu 2 vo fáze p, nech f je najvzdialenejší procesor vpravo od u ktorý vyšle u-čkové správy typu 2 vo fáze p a nech g je procesor s d(f,g) = $2^p - 1$. Potom pre ľubovoľný procesor y s f < y $\leq$ g, y nevysiela správy typu 2 vo fáze p.

Korektnosť 9 : Počet správ typu 2 vyslaných vo fáze p je najviac

$$n2^p/(2^{p-1} - 1).$$

- vyplýva z Lemy 2 a Lemy 8

Teoréma 10 : Úplný počet vyslaných správ je menej ako $\frac{3}{2}n \log n + \frac{8}{3}n$.

- Korektnosť 7 - algoritmus skončí počas najviac $\log n + 1$ fáz.
- Korektnosť 9 - počet vyslaných správ typu 2 je menej než $\frac{1}{2}n \log n + \frac{5}{3}n$.
- počet vyslaných správ typu 1 je najviac $n(\log n + 1)$

Počet bitov :

správy typu 1	$1 + \log n + m$ bitov	{m - veľkosť reprezentujúcich čísel}
správy typu 2	1 bit	{číslo fázy môže byť odstránené}

Modifikovaný algoritmus : $n \log n(\log n + m) + O(n \log n)$
Pôvodný algoritmus : $nm \log n + O(n \log n)$

Algoritmus

1. V 1. etape sú všetky procesory aktívne

2. {Nepárna etapa}
Aktívny procesor p vyšle <1, MAXID(p)>;
Ak aktívny procesor v prijme <1,i> potom :
Ak i > MAXID(v) potom v bude pasívny
inak choď do párnej etapy

3. {Párna etapa}
Aktívny procesor v vyšle <2, MAXID(v)>;
Ak aktívny procesor w prijme <2,j> potom :
Ak j < MAXID(w) potom w bude pasívny
inak MAXID(w) := j; choď do nepárnej etapy

3.4 Synchrónne kruhy

	Čas	Komunikácia
Frederickson, Lynch (1984)	$O(n2^i)$	$O(n)$
Vitányi (1984)	$O(n2^i)$	$O(n)$
Gafni (1985)	$O(n \log i)$	$O(n)$
Overmars, Santoro (1986)	$O(n \log n)$	$O(n)$

i - identifikačné číslo šéfa

Cieľ : zistiť šéfa s minimálnym identifikačným číslom

1° Vyšleme <i> rýchlosťou 2^i

2° Ak "rýchlejšia" správa predbehne "pomalšiu", zruší "pomalšiu" správu.

3° Ak <i> dorazí do procesora j (j<i), ten zruší správu.

4° Ak <i> dorazí do procesora i, tento procesor je šéf.

Ak procesory začnú výpočet v tom istom kroku, potom algoritmus vyšle dokopy 2n správ.

- minimálne <i> prejde dĺžku n
- ak j > i : j prejde dĺžku 1 , ako aj i prejde dĺžku 2^{j-i} súčasne

Preto j prechádza cestu dĺžky najviac $n/2^{j-i}$

3.5 Topológia všeobecnej siete

	Komunikácia	Čas
Gallager, Humblet, Spira (1981)	$O(m + n \log n)$	$O(n \log n)$
Chin, Ting (1985)	$O(m + n \log n)$	$O(n \log^* n)$
Gafni (1985)	$O(m + n \log n)$	$O(n \log^* n)$
Awerbuch (1987)	$O(m + n \log n)$	$O(n)$

Dolné ohraničenie $\Omega(m + n \log n)$ správ (najhorší a priemerný prípad)

- výraz $\Omega(n \log n)$ je dolné ohraničenie, pretože všeobecné siete zahŕňajú kruhy, pre ktoré $\Omega(n \log n)$ tvorí dolné ohraničenie

- výraz m je dolné ohraničenie :
  Predpokladajme, že algoritmus výberu A má výpočet C na sieti G s menej než m vymenenými správami. Skonštruujme sieť G' spojením dvoch kópií G.
  Obe časti siete G' majú to isté relatívne usporiadanie ako v G. Výpočet C môže byť simulovaný simultánne v oboch častiach G', uznaním výpočtu, v ktorom sa dva procesory stanú vybranými, čo je v spore s predpokladom, že výpočet nájde jediného šéfa.

3.6 Úlohy

Úloha 3.1 Ukážte, že existuje algoritmus pre problém výberu na stromoch používajúci O(n) správ a O(priemer) časových jednotiek.

Úloha 3.2 Ukážte, že existuje algoritmus pre problém výberu na hyperkockách používajúci O(nlogn) správ.

Úloha 3.3 Udajte synchrónny algoritmus výberu pre hyperkocky s zmyslom pre smer ktorý používa O(n) správ.

Kapitola 4

Minimálna kostra

Problém výberu
Problém kostry
} $\theta(m + n.\log n)$ zložitosť správ vo všeobecných sieťach

Callager, Humblet, Spira (83) $2.m + 5.n.\log n$ správ

GHS algoritmus závisí na nasledujúcich predpokladoch :

1. Každá hrana e má jedinečnú váhu $\omega(e)$ ($\omega(e)$ - reálne číslo)

2. Všetny uzly na začiatku spia. Niektoré sú zobudené spontánne, iné môžu prijať správu zatiaľ čo ešte stále spia.

Nech G=(V,E) je ohodnotený graf. Ohodnotenie kostry T grafu G sa rovná sume n-1 hrán v T. T je minimálna kostra (MST - minimal spanning tree), ak žiadna kostra grafu G nemá menšie ohodnotenie ako T.

L1: Ak sú ohodnotenia všetkých hrán rôzne, tak existuje jediná kostra.

Fragment - podgraf kostry

L2: Ak F je fragment a e je hrana vychádzajúca z F s najmenším ohodnotením, potom $F \cup \{e\}$ je fragment.

4.1 Sekvenčné algoritmy

Borůvka

Prim - začne s jediným fragmentom a zväčšuje ho o vychádzajúce hrany s najmenším ohodnotením.

Krushal - začne s množinou jednouzlových fragmentov a spája fragmenty pridávaním hrany s najmenším ohodnotením vychádzajúcej z nejakého fragmentu.

4.2 Popis algoritmu GHS

Popíšeme algoritmus GHS z hľadiska fragmentov.

1. Množina fragmentov je udržiavaná tak, že zjednotenie všetkých fragmentov obsahuje všetky uzly.
2. Na začiatku táto množina obsahuje každý uzol ako jednouzlový fragment.
3. Uzly vo fragmente spolupracujú pri hľadaní hrany vychádzajúcej z fragmentu s najmenším ohodnotením.
4. Keď je známa vychádzajúca hrana z fragmentu s najmenším ohodnotením, fragment bude spojený s iným fragmentom pridaním vychádzajúcej hrany v spolupráci s tým druhým fragmentom.
5. Algoritmus skončí, keď zostane len jeden fragment.

Efektívna implementácia algoritmu vyžaduje zopár poznámok a techník.

1. **Meno fragmentu** Každý fragment má meno, ktoré je známe všetkým procesorom v danom fragmente. Procesory testujú či je hrana vnútorná alebo vychádzajúca porovnávaním ich mien fragmentov.
2. **Spájanie veľkých a malých fragmentov** Pri spojení dvoch fragmentov sa zmení meno aspoň v jednom fragmente. Na udržanie efektívnosti týchto zmien stratégia spájania je založená na idee, že sa menší pripája do väčšieho prebratím mena väčšieho fragmentu.
3. **Úrovne fragmentov** Každý fragment má priradenú úroveň, na začiatku 0. Pri spojení fragmentov na tej istej úrovni má výsledný fragment nové meno, a jeho úroveň je o jedna vyššia. Nové meno je ohodnotenie hrany (jadrovej hrany - core edge) spájajúcej dva fragmenty. Ak sa fragment F_1 pripojí do F_2 s vyššou úrovňou, nový fragment $F_1 \cup F_2$ má úroveň F_2 a rovnaké meno ako F_2.

Ak sú tieto pravidlá spájania dodržané, počet zmien mena fragmentu alebo úrovne v procesore je najviac nlogn.

Zhrnutie stratégie spájania :

Fragment F s menom FN a úrovňou L je označovaný ako F=(FN,L), nech l_F označuje vychádzajúcu hranu z F s najmenším ohodnotením.

Pravidlo A. Ak l_F vedie do fragmentu F'=(FN',L') s L<L', F sa spája s F', po čom má nový fragment meno FN' a úroveň L'. Tieto nové hodnoty sú vyslané všetkým procesorom vo F.

Pravidlo B. Ak l_F vedie do fragmentu F'=(FN',L') s L=L' a $l_F = l_{F'}$, potom dva fragmenty sa spoja v nový fragment s úrovňou L+1 menom $\omega(l_F)$. Tieto nové hodnoty sú vyslané všetkým procesorom v F a F'.

Pravidlo C. Vo všetkých zvyšných prípadoch (t.j. L > L' alebo L=L' a $l_{F'} \neq l_F$) fragment F musí čakať pokiaľ sa neaplikuje pravidlo A alebo B

4.3 Algoritmus GHS

var	$state_p$	:	{sleep, find, found}
	$stach_p[q]$	:	{basic, branch, reject} pre každé $q \in Neigh_p$;
	$name_p$, $bestwt_p$	:	real;
	$level_p$	:	integer;
	$testch_p$, $bestch_p$, $father_p$	:	$Neigh_p$;
	rec_p	:	integer;

1. Ako prvý krok akcie každého procesora musí byť inicializovaný algoritmus :
   nech pq je kanál procesora p s najmenším ohodnotením;
   $stach_p[q]$:= branch; $level_p := 0$; $state_p$:= found; $rec_p := 0$;
   vyšli <connect,0> do q;

2. Po prijatí <connect, L> od q:
   if L < $level_p$
   then {Spájanie - Pravidlo A}
   $stach_p[q]$ = branch; send <initiate, $level_p$, $name_p$, $state_p$> to q
   else if $stach_p[q]$ = basic
   then {Pravidlo B} process the message later
   else {Pravidlo C} send <initiate, $level_p + 1$, $\omega(pq)$, find> to q

3. Po prijatí <initiate,L,F,S> od q:
   $level_p$:= L; $name_p$:= F; $state_p$:= S; $father_p$:=q;
   $bestch_p$:=udef, $bestwt_p := \infty$;
   forall $r \in Neigh_p : stach_p[r] = branch \wedge r \neq q$ do
   send <initiate,L,F,S> to r;
   if $state_p$ = find
   then $rec_p := 0$; test;

4. procedure test;
   if $\exists q \in Neigh_p : stach_p[q]$ = basic
   then $testch_p := q$ with $stach_p[q]$ = basic and $\omega(pq)$ minimal;
   send <test, $level_p$, $name_p$> to $testch_p$
   else $testch_p$:= udef; report

5. Po prijatí <test, L F> od q:
   if L > $level_p$
   then {Odpoveď musí počkať} process the message later
   else if $F = name_p$
   then {vnútorná hrana}
   if $stach_p[q]$ = basic
   then $stach_p[q]$:= rejected;
   if $q \neq testch_p$
   then send <reject> to q
   else test
   else send <accept> to q

6. Po prijatí <accept> od q:
   $testch_p$:= udef;
   if $\omega(pq) < bestwt_p$
   then $bestwt_p := \omega(pq)$; $bestch_p := q$;
   report;

7. Po prijatí <reject> od q:
   if $stach_p[q]$ = basic
   then $stach_p[q]$:= reject;
   test;

8. procedure report:
   if $rec_p = \#\{q | stach_p[q] = \text{branch} \wedge q \neq father_p\}$ and
   $testch_p$ = udef
   then $state_p$:= found; send <report, $bestwt_p$> to $father_p$

9. Po prijatí <report,ω> od q:
   if $q \neq father_p$
   then {odpoveď na inicializačnú správu}
   if $\omega < bestwt_p$
   then $bestwt_p := \omega$; $bestch_p := q$;
   $rec_p := rec_p + 1$; report
   else {pq je core edge}
   if $state_p$ = find
   then process this message later
   else if $\omega > bestwt_p$
   then changeroot
   else if $\omega = bestwt_p = \infty$ then stop

10. procedure changeroot:
    if $stach_p[bestch_p]$ = branch
    then send <changeroot> to $bestch_p$
    else send <connect, $level_p$> to $bestch_p$;
    $stach_p[bestch_p]$:= branch;

11. Po prijatí <changeroot> :
    changeroot;

4.4 Korektnosť a zložitosť

Veta : Algoritmus GHS korektne vyráta minimálnu kostru (MST) použitím najviac $5.n.\log n + 2.m$ správ.

Hrana cez ktorú fragment vysiela správu <connect,L> je hrana s najnižším ohodnotením vychádzajúca z daného fragmentu. Z tohto vyplýva, že minimálna kostra je vyrátaná korektne, aj pri čakaní indukovanom algoritmom.

Čakanie indukované algoritmom pre <report,ω> na core edge nevedie k uviaznutiu, pretože každý spojný uzol (core node) prijme hlásenia od všetkých svojich synov, po ktorom je správa spracovaná.

- Ak správa z $F_1 = (level_1, name_1)$ dorazí do uzla $F_2 = (level_2, name_2)$
  <connect, $level_1$> musí čakať ak $level_1 \geq level_2$ a žiadna <connect, $level_2$> nebola vyslaná,
  <test, $level_1$, $name_1$> musí čakať ak $level_1 > level_2$ {(2),(5)}
  Preto niektoré z nasledujúcich platí :

  1. $level_1 > level_2$
  2. $level_1 = level_2 \wedge \omega(l_{F_1}) > \omega(l_{F_2})$
  3. $level_1 = level_2 \wedge \omega(l_{F_1}) = \omega(l_{F_2})$ a F_2 stále hľadá.

  Preto sa nemôže vyskytnúť žiadny cyklus uviaznutia.

Každá hrana je vyradená najviac raz (vyžaduje dve správy) $\Rightarrow 2.m$
Na každej úrovni uzol - prijme najviac 1 initiate, 1 accept
- vyšle 1 report, 1 changeroot alebo connect, 1 test

Úplný počet správ je ohraničený $2.m + 5.n.\log_2 n$.

Kapitola 5

Traverzovacie algoritmy

Vstup: Distribuovaná sieť, počiatočný bod pre traverzovanie (prechádzanie)

Výstup: Kostra vytvorená prechádzaním uložená distribuovane v sieti

	Komunikácia	Čas
Čisté traverzovanie		
Chang (1982)	$O(m)$	$O(n)$
Prehľadávanie do šírky		
Cheung (1983)	$O(n^3)$	$O(n)$
Awerbuch (1984)	kn^2, $(1 < k < n)$	$O(n \log_k n)$
Frederickson (1985)	$O(n\sqrt{m})$	$O(n\sqrt{m})$
Awerbuch, Galleger (1985)	$O(m2^{\sqrt{\log n \log\log n}})$	$O(n2^{\sqrt{\log n \log\log n}})$
Cheung, Zhu (1987)	$O(n^2)$	$O(n^2)$
Prehľadávanie do hĺbky		
Cheung (1983)	$\underline{2m}$	$2m$
Awerbuch (1985)	$4m$	$4n - 2n_1$
Lekehman, Menaski (1987)	$4m - n$	$\underline{2n - 2}$

5.1 Čisté traverzovanie

Shout-and-echo

1° Iniciátor prechádzania (traverzovania) vyšle správu EXPLORER paralelne všetkým susedným uzlom.

2° Ak EXPLORER dorazí do nenavštíveného uzla, označí tento uzol ako navštívený a hranu ako preskúmanú. Okrem toho vyšle správy EXPLORER všetkým susedom o ktorých nevie, či boli navštívení.

3° Ak EXPLORER dorazí do navštíveného uzla, odpovie správou ECHO.

4° if všetky správy ECHO sa vrátia do uzla a uzol nie je iniciátor
then ECHO je vyslané po preskúmanej hrane
else if uzol je iniciátor
then ukonči prechádzanie

Analýza :

Čas	2priemer(G)
Komunikácia	2m správ
Priestor (v uzle)	O(n) bitov

Aplikácie :

- Algoitmus pre triedenie identifikačných čísel všetkých uzlov siete
- Algoritmus na nájdenie 2-súvislých komponentov v sieti

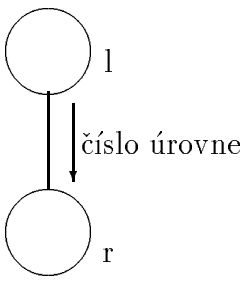
5.2 Prehľadávanie do šírky

(Breadth-first)

Cheung(1983)

1° Iniciátor prechádzania vyšle číslo úrovne 0 všetkým susedom.

2°



if r > l+1 then r:=l+1; vyšli l+1 susedom

3° Výpočet skončí : číslo úrovne uzla určuje vzdialenosť od iniciátora

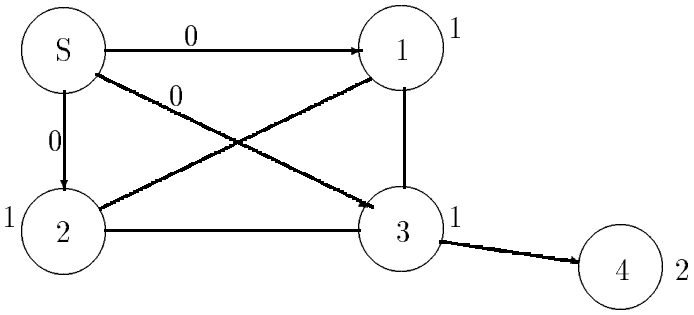
Analýza

Čas	$O(n)$
Komunikácia	$O(n^3)$ správ

Návestie uzla i sa mení i-krát
Všetky návestia v sieti sa menia celkove $n(n-1)/2$ krát
Pri zmene návestia sa vyšle n-2 správ

Celkove : O(n^3) správ Čas $O(n)$

Zhu, Cheung



Čas $O(n^2)$

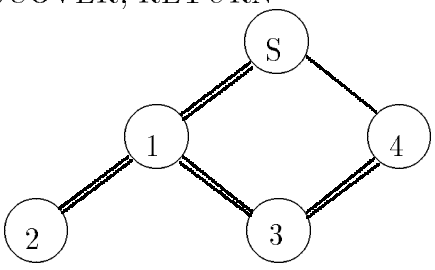
Komunikácia $O(n^2)$

5.3 Prehľadávanie do hĺbky

(DFS - Depth-first search)

Cheung 1989

komunikačná zložitosť 2m
časová zložitosť 2m
DISCOVER, RETURN



Averbuch (1985)

DISCOVER	presunie prechádzanie z navštíveného uzla do nenavštíveného uzla
VISITED	navštívený uzol informuje svojich usedov že bol navštívený
ACK	odpoveď na VISITED
RETURN	presunie prechádzanie z aktuálneho uzla do jeho otca v prechádzanom strome

Metóda : Navštívený uzol vyšle VISITED správy všetkým susedom okrem otca a čaká na všetky správy ACK ako odpovede na VISITED a potom vyšle správu DISCOVER nenavštívenému susedovi.

Analýza :

DISCOVER, RETURN	n-1 každý	} Spolu 4m správ
VISITED, ACK	2m-(n-1) každý	

Čas : $\left.\begin{array}{ll} 2 & \text{VISITED, ACK v uzloch stupňa} > 1 \Rightarrow 2n - 2n_1 \\ 2n - 2 & \text{DISCOVER, RETURN} \end{array}\right\} \Rightarrow$

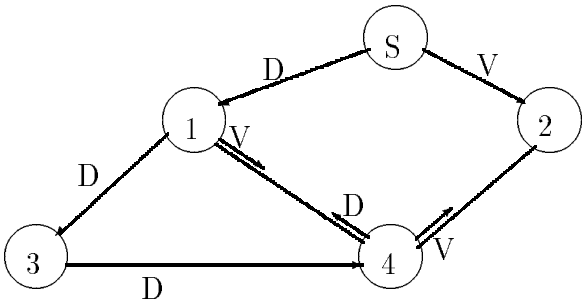
$\Rightarrow$ Spolu $4n - 2 - 2n_1$ (n_1- počet uzlov stupňa 1)

Algoritmus {pre uzol i}

```
var  Neighbors(i) :  množina susedov uzla i;
     Father(i) :     otec uzla i v DFS strome. Na začiatku, Father (i)=i;
     Unvisited(i) :  podmnožina Neighbors(i) zahŕňajúca susedov od ktorých ešte nebolo
                     prijaté VISITED. Na začiatku Unvisited(i) = Neighbors(i);
     flag(i) :       binárny flag, udržovaný pre každé j ∈ Neighbors(i), je rovný 1 v intervale
                     po vyslaní VISITED z i ku j a pred tým, ako bolo ACK prijaté späť.
                     Na začiatku flag(i,j) = 0.
{Inicializácia : Je vybraný uzol ako začiatočný bod prehľadávania. Na spustenie al-
goritmu si doručí sám sebe správu DISCOVERY.}
for  DISCOVER message from j do {i je navštívený prvý krát}
     Father(i) := j;
     for   all q∈ Neighbors except for node j do
           send VISITED message to q;
           flag(i,q) := 1;
     if    Neighbors(i) = j
     then  {j je jediný sused i}
           send RETURN to j;
for  RETURN message from q do
{v hľadaní sa pokračuje z uzla i ktorý už bol v minulosti navštívený}
     if    there exists k∈ Unvisited
     then  send DISCOVER to k;
           remove k from Unvisited(i)
     else  {všetci susedia boli navštívení}
           if Father(i)≠ i
           then send RETURN to Father(i) {backtracking}
           else STOP, the algorithm has terminated;
for  VISITED message from k do
     drop k from Unvisited(i);
     send ACK message to k;
for  ACK message from j do
     flag(i,j) := 0;
     if    flag(i,q) = 0 for all q∈ Neighbors(i)
     then  {pokračuj v hľadaní, majúc zozbierané všetky potvrdenia}
           deliver RETURN to itself;
```

Lakehman, Meenakshi 1987

DISCOVER, VISITED, RETURN



Čas 2n-2

Lakshmanan,Meenakshi,Thulasiraman 1987 Čas : 2n-2

Algoritmus {pre uzol i}

neighbors(i) :	množina susedov uzla i
father(i) :	otec uzla v DFS strome pre i
nomessage(i) :	množina susedov, o ktorých i nevie či už boli navštívení (žiadna správa VISITED, DISCOVER, RETURN)
visited(i)	= true, ak i bol navštívený
explore(i) :	sused, ktorým smerom bola vyslaná správa DISCOVER

```
for   VISITED from j do
      delete j from nomessage(i);
      execute procedure recover;
for   DISCOVER from j do
      delete j from nomessage(i);
      execute procedure recover;
      if    node i was visited
      then  do nothing
      else  visited(i) := true;
            father(i) := j;
            execute procedure shift-center-of-activity
            for all p∈ neighbors(i) and p≠ father(i) and p≠ explore(i) do
                  send VISITED to p
for   RETURN from j do
      delete j from nomessage(i);
      execute procedure shif-center-of-activity
```

procedure recover;

```
{inicializuj zotavenie z chyby, ak je to potrebné}
 if    explore(i) = j
 then  execute procedure shift-center-of-activity
 else  do nothing
```

```
procedure shift-center-of-activity;
    if    ∃ k ∈ nomessage(i)
    then  explore(i) := k;
          send DISCOVER to k;
    else  explore(i) := i;
          if    father(i) = i
          then  terminates computation {v uzle z ktorého sme začínali}
          else  send RETURN to father(i);
```

5.4 Traverzovanie torusu

$n \times n$ **torus-graf** je G=(V,E), kde V=$\mathbf{Z}_n \times \mathbf{Z}_n = \{(i,j)|0 \leq i,j < n\}$ a
E = $\{(i,j),(i',j')|(i = i' \wedge j = j' \pm 1) \vee (i = i' \pm 1 \wedge j = j')\}$.

Ak sa predpokladá, že torus má zmysel pre smer, t.j. v uzle (i,j) kanál do (i,j+1) je označený Up, kanál do (i,j-1) Down, kanál do (i+1,j) Right a kanál do (i-1,j) ako Left.

Znalosť topológie je ohraničená na značky kanálov.

Torus je Hamiltonovský graf, teda v toruse existuje Hamiltonovský cyklus.

Znak je vyslaný po takomto cykle použitím algoritmu Travtorus. Po k-tom kroku znaku je vyslaný nahor, ak n|k (n delí k), inak je vyslaný doprava.

Algoritmus Travtorus

```
for  the initiator, execute once:
     send <num,1> to Up;
for  each processor, upon receipt of the token <num,k>:
     if    k = n^2
     then  decide
     else  if n|k
           then send <num,k+1> to Up
           else send <num,k+1> to Right;
```

5.5 Traverzovanie hyprekociek

n-rozmerná hyperkocka je G=(V,E), kde V=$\{(b_0,\ldots,b_{n-1})|b_i = 0,1\}$
a E=$\{(b_0,...,b_{n-1})(c_0,..,c_{n-1})|$b a c sa líšia v jednom bite$\}$.

Predpokladá sa, že hyperkocka má zmysel pre smer, t.j. kanál medzi uzlami b a c, kde b a c sa líšia bitovým číslom i, je označený "i" z oboch strán.

Predpokladá sa, že značky uzlov nie sú známe procesorom, ich topologická znalosť je ohraničená na značky kanálov.

Hyperkocka je Hamiltonovská, a Hamiltonovský cyklus je prechádzaný použitím Travhypercube.

Algoritmus Travhypercube

for the initiator, execute once:
send <num,1> throung channel n-1;
for each processor, upon receipt of the token <num, k> :
if $k=2^n$
then decide
else {nech l je najväčšie číslo také, že $2^l|k$}
send <num,k+1> throung channel l;

5.6 Úlohy

Úloha 5.1 : Napíšte algoritmus, ktorý vyráta prefixovú značkovaciu schému pre všeobecnú sieť použitím 2m správ a času O(n). Môžete udať algoritmus, ktorý vyráta značkovaciu schému v čase O(priemer) ?

Úloha 5.2 : Udajte prechádzanie hyperkocky bez zmyslu pre smer ktoré používa O(nlogn) správ.

Úloha 5.3 : Ukážte, že DFS algoritmus s poznaním susedov prechádza všeobecnú sieť použitím 2n-2 správ za 2n-2 časových jednotiek.

Kapitola 6

Routovacie algoritmy

Routovanie je rozhodovacia procedúra, ktorou uzol vyberie jedného (alebo viac) z svojich susedov na vyslanie správy na jej cestu do cieľa. Úlohou je vygenerovať rozhodovaciu procedúru na vykonávanie tejto funkcie a zabezpečiť doručenie každej správy.

routovacia tabuľka - informácie o topológii siete uložené v uzle

Routovací problém

- Výpočet tabuľky (na začiatku alebo po zmene topológie)
- Doručovanie správ

Kritériá pre "dobré" routovacie metódy :

1. *Korektnosť* (algoritmus musí doručiť každý správu do jej cieľa)
2. *Zložitosť* (zložitosť správy, časová, pamäťová zložitosť výpočtu tabuľky)
3. *Efektívnosť* ("dobrá" routovacia cesta - krátka, zabezbečuje vysokú priepustnosť)
4. *Robustnosť* (v prípade zmeny topológie algoritmus upraví routovacie tabuľky)
5. *Prispôsobivosť* (algoritmus vyrovnáva zaťaženie kanálov a uzlov úpravami tabuliek)
6. *Spravodlivosť* (algoritmus musí poskytovať služby každému užívateľovi na tej istej úrovni)

"Najlepšia" routovacia cesta môže byť vzhľadom na :

1. *Minimum hop* (úsek, počet prejdených hrán)
2. *Najkratšia cesta* (suma ohodnotení na ceste)
3. *Minimálne zdržanie* (ako (2) pre dynamicky priraďované ohodnotenia hranám)

6.1 Algoritmus Netchange

Tajibnapis (1977) - vyrátava routovacie tabuľky, ktoré sú optimálne vzhľadom na "minimum-hop" a upravuje tabuľky s ohľadom na dynamické zmeny v topológii siete.

Algoritmus Netchange závisí na predpokladoch :

(N1) Uzly poznajú veľkosť siete (n).

(N2) Kanály spĺňajú fifo predpoklad (správy idú postupne za sebou).

(N3) Uzlom sa oznamujú chyby a opravy ich priľahlých kanálov.

(N4) Cena cesty je rovná počtu kanálov na tej ceste.

V každom uzle u: tabuľka $Nb_u[v]$ udáva pre každý cieľ v suseda uzla u.

Požiadavky algoritmu :

(R1) Ak topológia siete zostane konštantná po konečnom počte topologických zmien, potom algoritmus skončí po konečnom počte krokov.

(R2) Keď algoritmus skončí, tabuľka $Nb_u[v]$ spĺňa

(a) ak $u = v$, potom $Nb_u[v] =$ local;

(b) ak existuje cesta z u do $v \neq u$, potom $Nb_u[v] = w$, kde w je prvý sused u na najkratšej ceste z u do v;

(c) ak neexistuje žiadna cesta z u do v, potom $Nb_u[v] =$ udef.

Algoritmus NETCHANGE

var	$Neigh_u$	: množina uzlov;	{susedia uzla u}
	D_u	: pole 0..n;	{$D_u[v]$ určuje d(u,v)}
	Nb_u	: pole uzlov;	{$Nb_u[v]$ je is uprednostnený sused pre v}
	$ndis_u$	: pole 0..n;	{$ndis_u[w,v]$ určuje d(w,v)}

Inicializácia :

forall $w \in Neigh_u$, $v \in V$ do $ndist_u[w,v] :=$ n;

forall $v \in V$ do $D_u[v] :=$ n; $Nb_u[v] :=$ udef;

$D_u[u] := 0$; $Nb_u[u] :=$ local;

forall $w \in Neigh_u$ do send <mydist,u,0> to w

Procedúra Recompute(v) :

if $v = u$
then $D_u[v] := 0$; $Nb_u[v] :=$ local
else {urči vzdialenosť do v}
d:= 1+min$\{ndist_u[w, v] | w \in Neigh_u\}$
if d < n
then $D_u[v] :=$ d; $Nb_u[v] := w$ with $1 + ndis_u[w, v] =$d
else $D_u[v] :=$ n; $Nb_u[v] :=$ udef;
if $D_u[v]$ has changed
then forall $x \in Neigh_u$ do send <mylist, v, $D_u[v]$> to x

Spracovanie správy <mydist,v,d> od suseda w:

{<mydist,v,d> je na vrchu fronty Qwv}

receive <mydist,v,d> from w;

$ndist_u[w, v] :=$ d; Recompute(v);

Po chybe kanála uw:

receive <fail, v>; $Neigh_u := Neigh_u - \{w\}$;

forall $v \in V$ do Recompute(v);

Po oprave kanála uw :

receive <repair,w>; $Neigh_u := Neigh_u \cup \{w\}$;
forall $v \in V$ do
$ndis_u[w, v] :=$ n;
send <mydist, v, $D_u[v]$> to w;

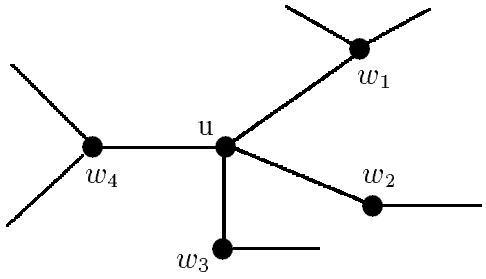
Veta : Ak topológia siete zostane konštantná po konečnom počte topologických prípadoch, potom algoritmus dosiahne stabilnú konfiguráciu po konečnom počte krokov.

Uzol nikdy nevie, či jeho routovacia tabuľka korektne zobrazuje topológiu siete a nemôže odkladať posielané správy pokiaľ nie je dosiahnutá stabilná konfigurácia.

Je možné priradiť ohodnotenie každému kanálu a upraviť algoritmus na vyrátanie najkratšej cesty namiesto cesty minimum-hop.

Bez topologických zmien, algorimus Netchange vymení $O(n^2m)$ správ na vyrátanie minimum-hop routovacích tabuliek pre problém najkratšej cesty pre všetky dvojice.

6.2 Routovanie s kompaktnými routovacími tabuľkami



Cieľ	Kanál
v_1	w_2
u	–
v_j	w_3
v_r	w_1

Kanál	Cieľ
w_1	$\ldots, v_n$
w_2	$v_1, \ldots$
w_3	$\ldots, v_j, \ldots$
w_4	

6.2.1 Schéma značenia stromu

(Tree-labeling scheme)

Santoro, Khatib(1985)

Cyklický interval [a,b) v $\mathbf{Z}_n(= \{0, 1, \ldots, n-1\})$ je množina definovaná

$$[a,b) = \begin{cases} \{a, a+1, \ldots, b\text{-}1\} & \text{ak } a < b \\ \{0, \ldots, b\text{-}1, a, \ldots, n\text{-}1\} & \text{ak } a \geq b \end{cases}$$

Cyklický interval [a,b) sa nazýva lineárny , ak a < b.

Veta : Uzly stromu T môžu byť očíslované takým spôsobom, že pre každý vychádzajúci kanál každého uzla množina cieľov, ktoré musia byť routované cez ten kanál je cyklický interval.

Vezmime si v_0 ako koreň stromu T, T[w] označuje podstrom stromu T s koreňom w.

Očíslujme uzly stromu T traverzovaním preorder, teda uzly v T[w] sú očíslované nejakým lineárnym intervalom, povedzme [a_w,b_w).
Uzol w pošle synovi správu s cieľom z [a_w,b_w).
Uzol w pošle správu svojmu otcovi s cieľom zo $\mathbf{Z}_n - [a_w, b_w)$.

Algoritmus posielania intervalov

{Správa s adresou d bola prijatá alebo generovaná v uzle u}
{Predpokladajme, že kanály u stupňa deg sú označené ako $\alpha_1, \ldots, \alpha_{deg}$,
kde $\alpha_1 < \ldots < \alpha_{deg}$}
ak d = label(u)
potom spracuj správu lokálne
inak vyber α_i takú, že $d \in [\alpha_i, \alpha_{i+1})$;
vyšli správu cez kanál označený ako α_i

Schéma značkovania stromu na stromoch routuje optimálne.

Na routovanie nestromových sietí zvoľte pevnú kostru siete.

Lema : Neexistuje uniformné ohraničenie pomeru $d_T(u,v)$ a $d_G(u,v)$.
Toto platí aj v špeciálnom prípade hop-miery kanálov.

Vezmime si kruh, kostra kruhu neobsahuje jeden kanál, povedzme xy.
Teraz $d_G(x,y) = 1$, $d_T(n,y) = n-1$.

Lema : T môže byť zvolená tak, že pre všetky u,v : $d_T(u,v) \leq 2D_G$

Nevýhody schémy routovania stromu :

1. Kanály nepatriace T sú nevyužité, čo je plytvanie zdrojmi siete.
2. Premávka je koncentrovaná na strom, čo môže viesť k zahlteniu.
3. Každé jediné porušenie kanálu z T rozdeluje sieť.

6.3 Intervalové routovanie

van Leeuven, Tan(1987)

Intervalová značkovacia schéma (interval-labeling scheme ILS) pre sieť je

1. priradenie rôznych značiek zo Z_n uzlom siete;
2. pre každý uzol, priradenie rôznych značiek zo Z_n kanálom tohto uzla.

Algoritmus intervalového značkovania pre danú ILS posiela správy ako routovací algoritmus značkovania na strome.

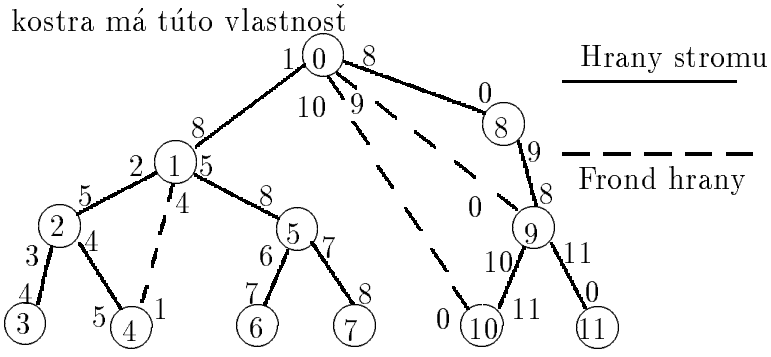
ILS je prijateľná, ak všetky správy posielané týmto smerom môžu dosiahnuť svoj cieľ.

Veta : Pre každú súvislú sieť G existuje prijateľná ILS.

Majme kostru, frond hrana je hrana, ktorá nepatrí tejto kostre. v je predchodca u $\Leftrightarrow u \in T[v]$.

Lema : Existuje kostra taká, že všetky frond hrany sú medzi uzlom a prechodcom tohto uzla.

DSF kostra má túto vlastnosť



DFS ILS pre G (s ohľadom na T) je :

1. Značky uzlov v podstrome T[w] sú značené číslami z $[l_w, l_w + |T[w]|]$
   Zapísať $k_w = l_w + |T[w]|$.
2. Značka hrany uw pri uzle u je α_{uw}.
   (a) Ak uw je frond hrana potom $\alpha_{uw} = l_w$
   (b) Ak w je syn u (v T) potom $\alpha_{uw} = l_w$
   (c) Ak w je otec u potom $\alpha_{uw} = k_u$ pokiaľ nie je k_u = n a u nemá frond spojenie do koreňa.
   (V tej druhej situácii, frond hrana je označená 0 pri u podľa pravidla (a), teda priradenie značky k_u môže ohroziť požiadavku, že všetky značky hrán v u sú rôzne)
   (d) Ak w je otec u, u má frond spojenie na koreň, a k_u =n, potom $\alpha_{uw} = l_w$.

Pre každú sieť existuje platná (validná) ILS, ale ILS prehľadávaním do hĺbky nie je optimálna.

ILS je optimálna, ak doručí všetky správy cez optimálne cesty.

ILS je susedská (neighborly), ak doručí správu z jedného uzla do suseda jedným prechodom (hop).

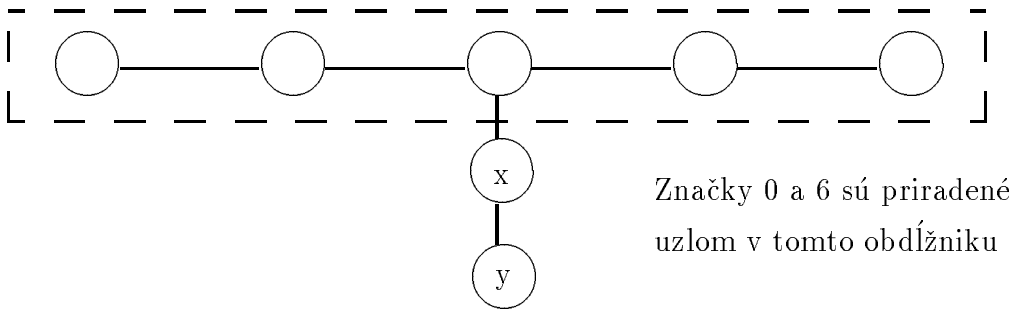
LS je lineárna, ak interval priradený každej hrane je lineárny.

6.3.1 Efektívnosť intervalového routovania

Teoréma : Existuje sieť G taká, že pre každú ILS pre G existujú uzly u,v, také, že správa z u do v je doručená len po $\frac{7}{4}D_G$ prechodoch hranami (hops).

Teoréma : Existuje sieť, pre ktorú neexistuje žiadna lineárna ILS.

- Graf - pavúk , s tromi nohami



- Najmenšia značka (0) a najväčšia značka (6) sú priradené najviac dvom nohám.
  Nech x je prvý uzol z centra nohy neobsahujúcej značky 0,6.
  Uzol x posiela správy pre 0 a 6 smerom do centra, a jediný lineárny interval,

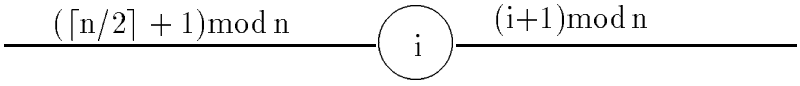
čo obsahuje aj 0 aj 6 je celá množina Z_n. Následne, x teda posela aj správy pre y smerom na centrum, a tie nikdy nedosiahnu ich cieľ.

6.3.2 Optimálnosť intervalového routovania : špeciálne topológie

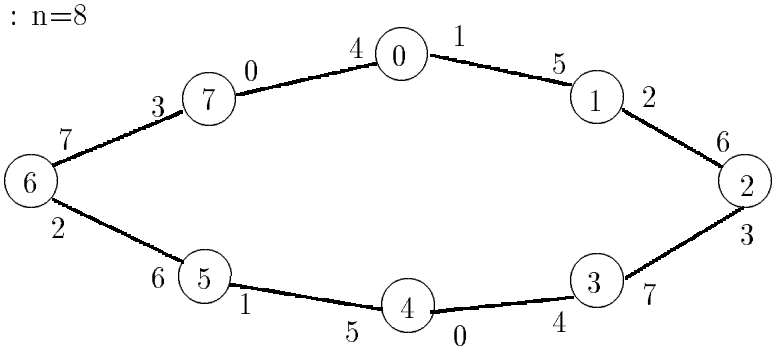
Stromy DSF ILS je minimum-hop

Kompletné grafy DSF ILS je minimum-hop

Kruhy (orientácia v smere hod. ručičiek v kruhu)

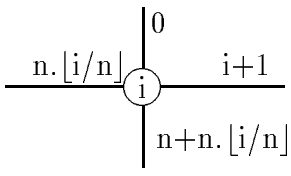


Príklad : n=8



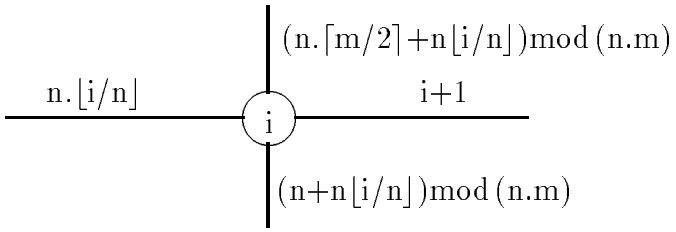
Siete n × m (n riadkov, m stĺpcov)

- Značky uzlov sú priraďované v poradí podľa riadkov

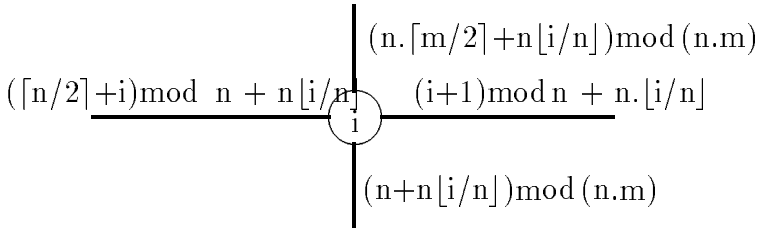


- Značky kanálov :
  - kanál-hore : 0
  - kanál-dolu : značka najľavejšieho uzla v nasledujúcom riadku
  - kanál-vľavo : značka najľavejšieho uzla v tom istom riadku
  - kanál-vpravo : značka pravého susedného uzla

Siete s cyklickými stĺpcami (n riadkov, m stĺpcov)

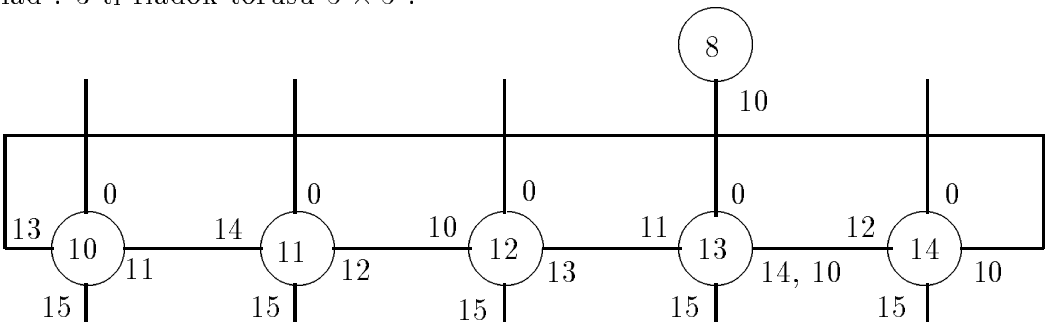


Torus (Siete s cyklickými riadkami aj stĺpcami)
Optimálna ILS s 2-značkami



Pre nejaké i (i $\neq$ r.n pre r= $0, 1, \ldots, m-1$) ak horizontálny kanál neobsahuje r.n, potom pridaj r.n značku kanálu s najvyššou hodnotou značky.

Príklad : 3-tí riadok torusu 5×5 :



$13 \mapsto 10$

Hyperkocka existuje minimum-hop lineárna ILS

Outerplanárne siete existuje ILS vzhľadom na najkratšiu cestu pre o. siete so všeobecnými ohodnoteniami kanálov

Výhody intervalového routovania oproti klasickému routovaniu

1. Nízka priestorová zložitosť — Routovacie tabuľky potrebujú $O(\deg . \log n)$ bitov
2. Efektívnosť výpočtu routovacích tabuliek — Routovacie tabuľky pre DFS ILS môžu byť vzrátané distribuovaným DFS, ktorý potrebuje $O(m)$ správ a čas $O(n)$.

3. Optimálnosť — Routovacia metóda je schopná zvoliť optimálnu cestu pre rôzne triedy sietí

Aplikácie : V transputeroch - Inmos C104 routovacie čipy

Nevýhody intervalového routovania

1. *Slabá stabilita* (nedostatočná robustnosť) : Malá zmena topológie môže vyžadovať kompletné prerátanie routovacích tabuliek, vrátane priradení nových značiek každému uzlu.
2. *Neoptimálnosť* : DFS ILS môže routovať správy cez cesty siete dĺžky $\Omega(n)$, dokonca aj v prípade siete malého polomeru.

6.4 Prefixové routovanie

Bakker, van Leeuwen, Tan (1993)

- Routovacie tabuľky môžu byť vyrátavané použitím kostry (nielen DFS). Umožňuje pridávanie kanálov a uzlov. Optimálnosť nôže byť zlepšená použitím kostry nízkej hĺbky.
- Prefixová značkovacia schéma (prefix labeling scheme) (nad Σ) pre sieť G je
  1. priradenie rôznych reťazcov z Σ^* uzlom z G,
  2. pre každý uzol, priradenie rôznych reťazcov kanálom tohto uzla.
- Prefixové posielanie (pre uzol u)
  Algoritmus
  {Správa s adresou d bola prijatá alebo vygenerovaná v uzle u}
  ak d=l_u
  potom doruč správu lokálne
  inak α_i := najdlhšia značka kanálu taká, že $\alpha_i \triangleleft$ d; (α_i je prefix d)
  vyšli správu kanálom označeným α_i
- PLS je platná (validná), ak všetky správy posielané týmto spôsobom môžu dosiahnuť svoj cieľ.

Teoréma : Pre každú súvislú sieť G existuje platná PLS.

- Strom PLS pre G (s ohľadom na T) je prefixová značkovacia schéma v ktorej sú splnené nasledujúce pravidlá
  1. Značka koreňa je ε .
  2. Ak w je syn u, potom l_w rozšíri l_u o jedno písmeno, teda ak $u_1, \ldots, u_k$ sú synovia vrchola u z T, potom $l_{u_i} = l_u.a_i$, kde $a_1, \ldots, a_k$ sú rôzne písmená z Σ.

3. Ak uw je frond, potom $\alpha_{uw} = l_w$.
4. Ak w je syn u, potom $\alpha_{uw} = l_w$.
5. Ak w je otec u, potom $\alpha_{uw} = \varepsilon$ pokiaľ u nemá frond do koreňa, v takom prípade $\alpha_{uw} = l_w$

- Pre každú sieť G s priemerom D_G (rátané v hopoch) existuje prefixová značkovacia schéma taká, že doručí všetky správy v najviac $2.D_G$ hopoch (krokoch, skokoch).
- Priestorové požiadavky :
  Routovacia tabuľka uzla u potrebuje $O(\triangle.\delta.\log k)$ bitov
  kde $\triangle$ - stupeň uzla u
  δ - hĺbka T
  k - maximálny počet synov v T

6.5 Hierarchické routovanie

van Leeuwen, Tan (1986)

V hierarchickom routovaní je sieť rozdelená na skupiny (clustery), pričom každá skupina je súvislá podmnožina uzlov. V každej skupine existuje centrálny uzol ktorý môže vysielať správy do ostatných skupín. Každá skupina môže byť rozdelená na podskupiny, aby sa dosiahlo viacúrovňového rozdelenia siete.

Niektoré cenové parametre routovacej metódy závisia viac na veľkosti celkovej siete, než na dĺžke zvolenej cesty :

1. Dĺžka adresy : ILS - $O(\log n)$ IPS - dokonca viac
2. Veľkosť routovacej tabuľky : Kompaktné - $O(\deg)$, Nekompaktné - $O(n)$
3. Cena vyhľadávaní v tabuľke : závisí od veľkosti a počtu prístupov

Redukcia počtu routovacích rozhodnutí

Vhodným rozdelením siete na spojené súvislé skupiny

Lema : Pre každé $s \leq n$ existuje rozdelenie siete na skupiny $C_1, \ldots, C_m$ také, že

1. každá skupina je súvislý podgraf,
2. každá skupina obsahuje najviac s uzlov,
3. každá skupina má polomer najviac 2s.

$D_1, \ldots, D_m$- maximálna množina disjunktných súvislých podrafov, $|D_i| \geq s$, D_i má polomer $\leq s$.
Uzol $\notin \cup D_i$: spojenie do skupín najbližších k nemu cestou dĺžky $\leq$ s.
Dostaneme $C_1, \ldots, C_m$ spĺňajúce (1) + (2) + (3).

Teoréma : Pre každú sieť o n uzloch existuje routovacia metóda vyžadujúca najviac $O(\sqrt{n})$ routovacích rozhodnutí pre každú správu, a používa 3 farby.

Predpokladajme rozdelenie $C_1, \ldots, C_m$ podľa predchádzajúcej lemy. C_i obsahuje centrum c_i také, že $d(v, c_i) \leq 2s$ pre každé $v \in C_i$.

T je najmenší podstrom G spájajúci všetky centrá c_i. T obsahuje najviac m listov, teda najviac m-2 bodov vetvenia (uzly stupňa > 2).
Odkazujeme na uzly T ako na centrá, body vetvenia, uzly cesty, (zvyšné uzly).

Routovacia metóda : (priradenie farby každej správe)

- z počiatočného uzla do centra c_i danej skupiny zdroja - ZELENÁ FÁZA
- z c_i cez T do centra c_j skupiny miesta určenia - MODRÁ FÁZA
- z c_j v C_j do cieľa - ČERVENÁ FÁZA

Implementácia :

ZELENÁ FÁZA	- pevný strom pre centrum každej skupiny - žiadne routovacie rozhodnutia
MODRÁ FÁZA	- uzly cesty - žiadne routovacie rozhodnutia
	- body vetvenia a centrá - robia routovacie rozhodnutia
ČERVENÁ FÁZA	- stratégia najkratšej cesty - robí najviac 2s rozhodnutí

Výpočet : $2m - 2 + 2s$, čo je najviac $2\frac{n}{s} - 2 + 2s$.
Zvolením $s \approx \sqrt{n}$ dáva ohraničenie $O(\sqrt{n})$.

Teoréma : Pre každú sieť o n uzloch a každé kladné celé číslo $f \leq \log n$ existuje routovacia metóda, ktorá vyžaduje najviac $O(fn^{1/f})$ routovacích rozhodnutí pre každú správu a používa 2f + 1 farieb.

- Konštrukcia z prechádzajúcej teorémy sa rekurzívne aplikuje na T. Skupiny majú rovnakú veľkosť s.
- Rozdelenie na skupiny sa opakuje t krát. Sieť G má n uzlov.
  Po jednom rozdelení na skupiny : Strom má n/s centier , n/s bodov vetvenia $\Rightarrow n.2s$ dôležitých uzlov .
  Po i rozdeleniach skupín na podskupiny : m_{i-1}/s centier, m_{i-1}/s bodov vetvenia
  (m_{i-1} dôležitých uzlov po i-1 rozdeleniach na podskupiny)
  Po f rozdeleniach skupín na podskupiny : najviac $m_f = n.(2/s)^f$ dôležitých uzlov
- každá úroveň rozdelenia zvýši počet farieb o dve : spolu 2f+1 farieb.
- Počet rozhodnutí :
  - V najvyššej úrovni : najviac 2mf rozhodnutí
  - V každej úrovni v skupine doručenia : s rozhodnutí

  Spolu : $2mf + sf$, zvolením $s \approx 2n^{1/f}$ dáva $mf = O(1)$.
  Preto $fs = O(f.n^{1/f})$

- Použitie približne $\log_2 n$ farieb vedie ku routovacej metóde ktorá vyžaduje $O(log_2 n)$ routovacích rozhodnutí. Zistenie farby správy je tiež druh routovacieho rozhodnutia, ale zahŕňa malé tabuľky (najviac dĺžky $O(log_2 n)$) a je použité len v malých častiach uzlov.

6.6 Úlohy

Úloha 6.1 Existuje ILS ktorá nevyužíva všetky kanály na routovanie? Existuje taká platná? Optimálna?

Úloha 6.2 Ukáž ILS hľadaním do hĺbky pre kruh o n uzloch. Nájdi uzly u a v také, že d(u,v) = 2 , a schéma používa n-2 prechodov po hranách (hops) na doručenie správy z u do v.

Úloha 6.3 Nájdi algoitmus na určenie, či ILS na n uzlovej sieti je platná, ktorý používa $O(n^2)$ výmen správ.

Úloha 6.4 Existuje minimum-hop platná lineárna značkovacia schéma na

(i) kruhoch veľkosti > 4?

(ii) všeobecných stromoch?

*** Úloha 6.5** Ukáž minimum-hop platnú (validnú) prefixovú značkovaciu schému na hyperkockách.

Kapitola 7

Problém dohody v nespoľahlivých systémoch

7.1 Výpočtové modely

Distribuovaná sieť - n procesorov, m kanálov, nanajvýš t chybných procesorov

Protokol je t-odolný (t-resilient, odolný voči t-chybám), ak pracuje správne tak dlho, kým nie je počas vykonávania viac ako t chybných procesorov.

Dva druhy chýb procesora:

Havária (Crash) - procesor zastaví všetky činnosti

Byzantská chyba (Byzantine failure) - žiadne domienky o správaní sa chyby procesora (môže to byť "odporujúce" protokolu)

Predpoklad - systém správ je úplne spoľahlivý

- každý procesor môže spoľahlivo zistiť odosielateľa správy, ktorú prijal
- sieť je úplný súvislý graf

Môže byť zistená chyba procesora ?

ÁNO : - v modele s presnými hodinami a ohraničeniami časov prenosu správ
- v synchrónnom modele

NIE : - v plne asynchrónnom modele

Typy správ:

- Nepodpísané správy (model nepodporuje podpisy)
- Podpísané správy (autentifikačné protokoly používajú podpísané správy)
  a) podpisy nemôžu byť sfalšované chybujúcim procesorom
  b) môže byť určená zmena obsahu správy

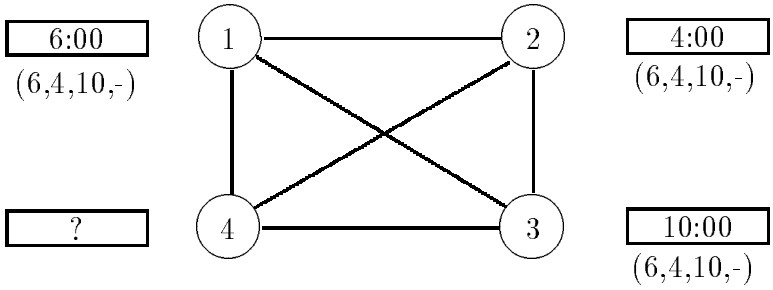
c) môže byť overená autentifikácia podpisu

Príklady:

4 procesory v plne synchrónnej sieti, jeden z nich "Byzantínsky". Každý procesor má interné hodiny.

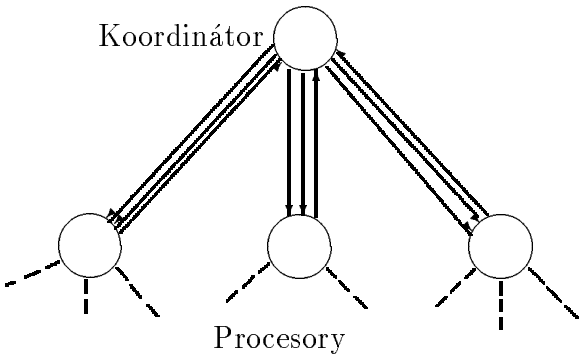
Problém: Nastaviť hodiny správnych procesorov na rovnaký "globálny" čas.

- Nie je dostatočná vzájomná výmena "interných" časov.
- Davis, Wakerley (1978) - vplyv chybujúcich procesorov môže byť odstránený viackolovými "hlasovacími schémami"



Dvojfázové odovzdávanie (Commit)

1. Koordinátor posiela "žiadosť o transakciu" všetkým procesorom.
2. Procesor posiela odpoveď na "žiadosť o transakciu" koordinátorovi.
3. Koordinátor na základe odpovedí urobí rozhodnutie a pošle ho procesorom



7.2 Problém dohody

(Consensus problem)

Procesory $p_1, p_2, \ldots, p_n$ nanajvýš t chybných

Každý p_i na začiatku obsahuje vstupnú hodnotu $x_i \in \{0, 1\}$

Problém: Každý p_i počíta výstupnú hodnotu $y_i \in \{0, 1\}$ takú, že

1. Dohoda: Ak p_i, p_j sú korektné procesory, potom $y_i = y_j$
2. Podmienka závislosti:
   - *Netriviálnosť:* pre každý procesor $y \in \{0,1\}$ existujú nejaké počiatočné $x_1, \ldots, x_n$ a nejaké prípustné vykonanie protokolu takého, že ak p_i je správny, potom $y_i = y$.
   - *Slabá zhoda:* Ak $x_i = x \in \{0,1\}$ pre všetky i, potom $y_i = x$ za predpokladu, že počas činnosti protokolu sa skutočne nevyskytnú žiadné chyby.
   - *Silná zhoda:* Ak $x_1 = \ldots = x_n, x \in \{0,1\}$, potom $y_i = x$ pre každý správny (korektný) procesor p_i.

7.3 Problém interaktínej konzistencie

(Interactivity consistency problem)

Každý p_i začiatočne obsahuje vstupnú hodnotu $x_i \in \{0,1\}$

Problém je pre každý p_i vypočítať vektor konsenzu (dohody) $v_i = (u_1^i, \ldots, u_n^i)$, $u_j^i \in \{0,1\}$ taký, že

1. Ak p_i, p_j sú správne procesory, potom $v_i = v_j$
2. **Slabá verzia:** Pre každý p_j, $u_i^j = x_i$ za predpokladu, že počas činnosti protokolu sa skutočne nevyskytnú žiadné chyby.
   Silná verzia: Ak p_j je korektný potom $u_i^j = x_i$.

7.4 Problém generálov

Generals problem

Problém spoľahlivého vysielania (Reliable broadcast problem)

Vyznačený procesor (generál, alebo vysielač) posiela svoj inicializačný bit x všetkým ostatným procesorom.

Podmienka dohody: Všetky spoľahlivé procesory musia dosiahnúť dohodu na výstupný bit (označený ako y).

Podmienka závislosti:

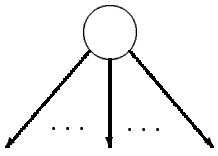
Slabá verzia: Hodnota dohody je $y = x$, ak sa počas činnosti protokolu nevyskytnú žiadne chyby.

Silná verzia: Hodnota po dohonutí je $y = x$, ak je generál správny.

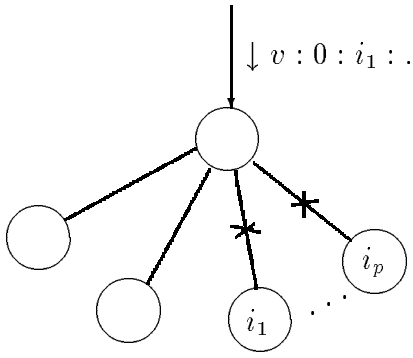
7.5 Vzťahy medzi problémami dohody.

- Problém generálov je špeciálny prípad problému interaktívnej konzistencie.
- Obrátene, n kópii generálovho algoritmu paralelne rieši IC problém
- IC algoritmus rieši problém konsenzu (hodnota konsenzu = väčšinová hodnota v konsenzus vektore.)
- Algoritmus dohody (Consensus alghoritm) + jedno kolo výmeny informácie rieši problém generálov.
  1. Generál posiela svoju hodnotu každému z ostatných procesorov.
  2. Všetky procesory (s inicializačnou hodnotou od generála) bežia dohodový algoritmus.
- Analogické výsledky platia aj pre slabé verzie problémov dohody.
- Problém využitia slabých verzií pri riešení silných verzií problémov dohody je otvorený problém.

1° Generál 0



Posiela v:0
(Ak je chybujúci, správa môže byť rôzna)



2° Procesor j

? platí
- prvý podpíše 0
- rôzne podpisy
- p<t

Ulož v vo V_j; vyšli $v : 0 : i_1 : \ldots : i_p : j$

3° Ak procesor p prijal iba hodnoty v potom v je hodnota konsenzu, ináč NIL.

7.6 Riešiteľnosť problémov dohody

Máme obmedziť problém generálov.

Veta 1 (Pease, Shostak, Lamport, 1980) $(t \leq n - 2)$

Existuje t-odolný autentifikovaný synchrónny protokol, ktorý rieši silný (slabý) Problém Byzantínskych generálov.

algoritmus SM(t);

{Počiatočne, $V_i = \emptyset$}

1. Generál (procesor p_0) podpíše a pošle svoju hodnotu (v:0) všetkým ostatným procesorom.
2. Pre všetky i:
   (a) Ak p_i prijíma v:0, potom
      i. $V_i = \{v\}$
      ii. posiela $v : 0 : i$ všetkým ostatným procesorom
   (b) Ak p_i prijíma $v : 0 : j_1 : \ldots : j_k$
      i. $V_i = V_i \cup \{v\}$
      ii. ak k < t
      potom posiela $v : 0 : j_1 : \ldots : j_k : i$ všetkým procesorom rôzným od $j_1, \ldots, j_k$
3. Pre všetky i :
   Ak V_i je jednoprvková množina, potom táto hodnota je vybraná ako hodnota konsenzu, ináč je vybraná fixná hodnota NIL

Zložitosť: t+1 kôl, $(n-1)(n-2)\ldots(n-t-1)$ správ

Dôkaz korektnosti:

$$\boxed{p_i, p_j \text{ sú správne} \Rightarrow V_i = V_j}$$

1° generál p_0 je korektný s hodnotou v, potom $V_i = V_j = \{v\}$

2° generál je chybný:
Nech p_i pozná v z $v : m_1 : \ldots : m_k$

(i) k < t+1, p_j nepozná v:
Potom p_j prijíma v od p_i v ďalšom kole.

(ii) k = t+1
Potom $p_{m_1}, \ldots, p_{m_t}$ sú chybné (ináč by p_i mal poznať v)
Preto p_{m_t+1} je správny a p_j pozná v v tom istom kole ako p_i.

Veta 2 Existuje t-odolný autentifikačný synchrónny protokol, ktorý rieši silný (slabý) Problém Byzantínskych generálov v ľubovoľnom grafe, v ktorom najmenší podgraf obsahujúci všetky správne (korektné) procesory je súvislý.

Riešenie je algoritmus SM(n-2)

Existuje algoritmus na t+d kôl, pričom d je priemer podgrafu.

Veta 3 (Lamport, Shostak, Pease, 1982) (t<n/3)
Existuje t-odolný synchrónny protokol bez podpisovania, ktorý rieši silný (slabý) Byzantínsky problém $\Leftrightarrow t < n/3$

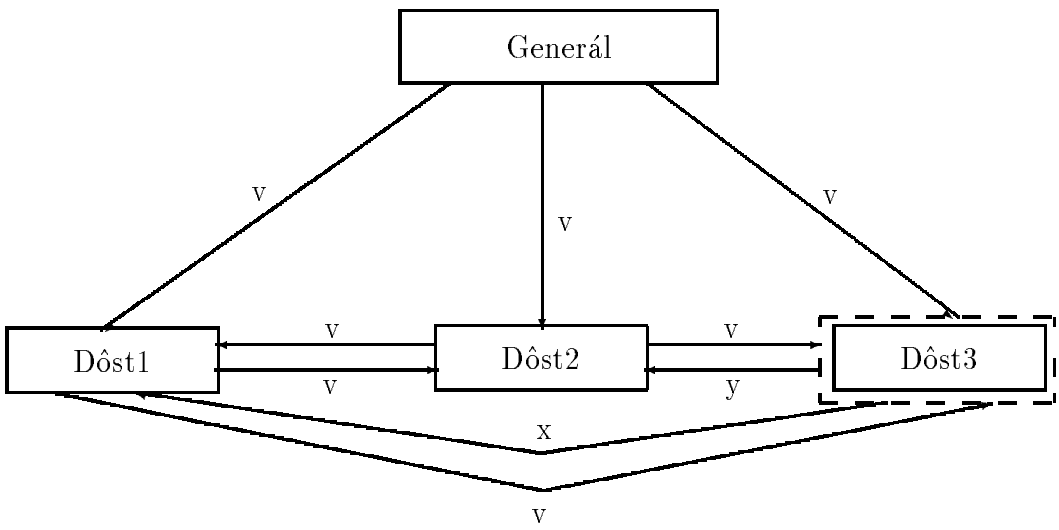
algoritmus OM(0)

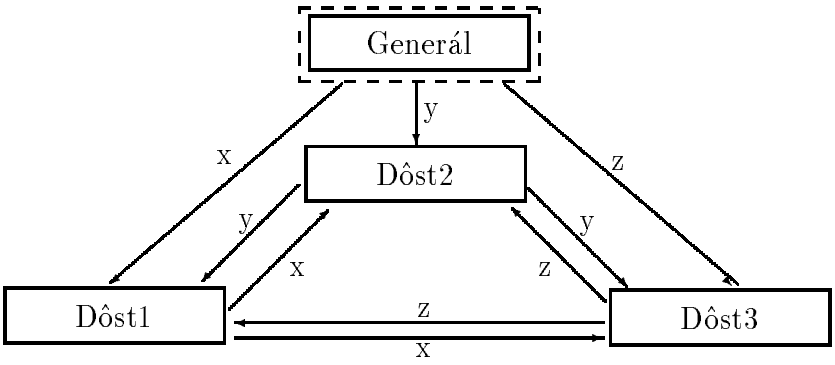
1. Generál posiela svoju hodnotu všetkým ostatným procesorom;
2. Každý procesor vyberie hodnotu, ktorú dostal od generála (alebo NIL, ak neprišla žiadna hodnota.)

algoritmus OM(t), t>0;

1. Generál posiela svoju hodnotu všetkým procesorom
2. Pre všetky $i \in < 1, n-1 >$:
   Nech v_i je hodnota, ktorú p_i prijalo od generála, alebo NIL.
   Potom p_i bude generál v OM(t-1) a vyšle svoju hodnotu všetkým procesorom
3. Pre i,j ($i \neq j$):
   Nech v_j je hodnota, ktorú p_i prijal od p_j v kroku (2) algoritmu OM(t-1), alebo NIL. Potom p_i vyberie MAJORITY$(v_1, \ldots, v_{n-1})$

MAJORITY$(v_1, \ldots, v_{n-1}) \equiv$ najčastejšie sa vyskytujúca hodnota v $\{v_1, \ldots, v_{n-1}\}$, inak NIL.





Korektnosť algoritmu OM

Lema 1 Pre ľubovoľné m,k, algoritmus OM(m) spĺňa podmienku (1) definície Byzantínskeho problému, keď nie je viac ako 2k+m procesorov a nanajvýš k chybných.

Indukciou na m:

m = 0: systém správ je spoľahlivý, generál je správny ⇒ algoritmus je pre m = 0 správny.

predpokladajme, že platí pre m-1:

m: krok (1) ↦ správny generál posiela v všetkým n-1 procesorom.
krok (2) ↦ správny procesor volá OM(m-1) s n-2 procesormi
Predpokladajme $n > 2k + m \Rightarrow n - 1 > 2k + (m + 1) \Rightarrow$ správny procesor j vyberie v_j = v; ako $n - 1 > 2k$ {väčšina správnych procesorov} ⇒ majority$(v_1, \ldots, v_{n-1})$ = v.

Lema 2 Pre ľubovoľné m algoritmus OM(m) spĺňa podmienky (1), (2) definície Byzantínskeho problému, ak je viac ako 3m procesorov a nanajvýš m chybných.

Indukciou na m:

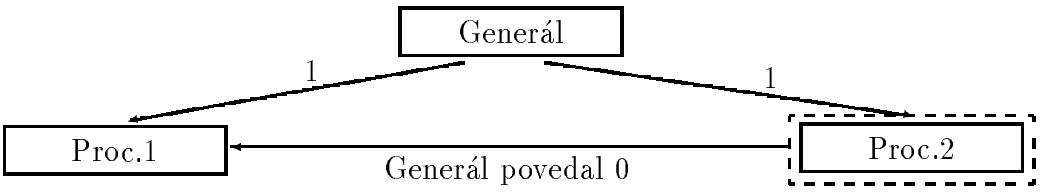
- 0 chýb: OM(0) spĺňa (1), (2)
- predpokladajme, že L2 platí pre m-1:

m:
- **korektný generál:** L1 (pre k=m) ⇒ OM(m) spĺňa (1)
  pre korektného generála ⇒ z (1) vyplýva (2)
- **chybný generál** + (m-1) chybných procesorov: Platí 3m-1 > 3(m-1) ⇒ (indukčný predpoklad) OM(m-1) spĺňa (1), (2) ⇒ z kroku (3) algorit. OM: ∀ i,j správne procesory: $v_i = v_j \Rightarrow$ OM spĺňa (2)

Neexistencia argumentu pre t < n/3

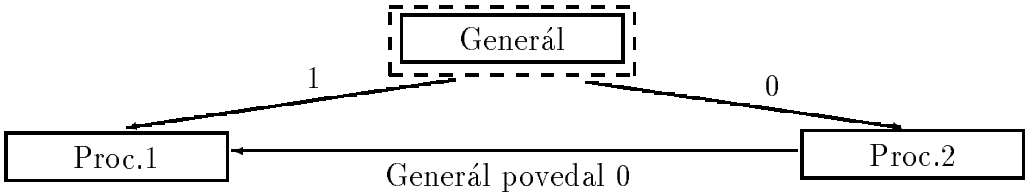
Neexistuje algoritmus pre n = 3, t = 1

Scenár I:



Procesor1 musí vybrať 1

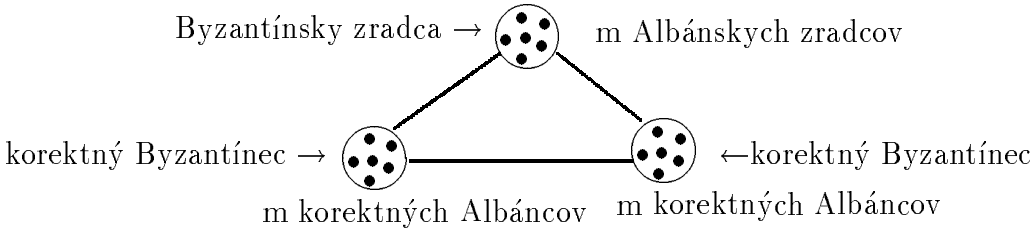
Scenár II:



II: Procesor 1 nerozlišuje oba prípady, vyberá 1
Procesor 2 spĺňa 0
$\Rightarrow$
$\Rightarrow$ spor s požiadavkou, že korektné procesory vyberú rovnakú hodnotu!

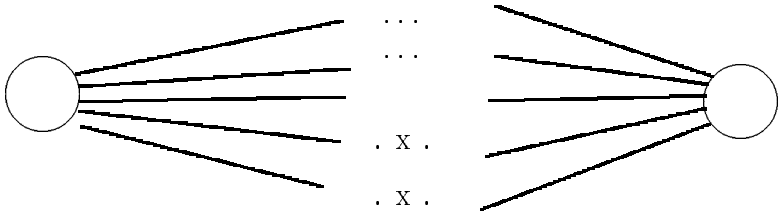
Neexistuje algoritmus pre n = 3m, t = m

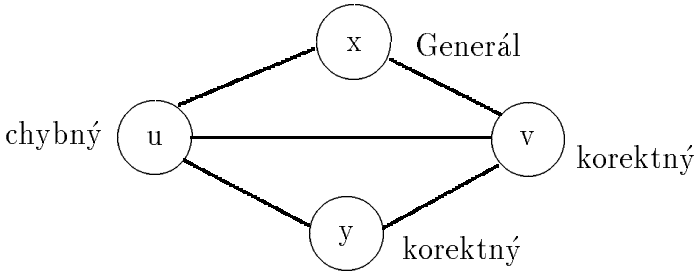
Predpokladajme, že taký algoritmus existuje.
3m - Albánskych dôstojníkov, m - zradcov



Korektný Byzantínsky dôstojník vyberie hodnotu konsenzu skupiny korektných Albáncov.

Veta 4 Uvažujme synchrónnu sieť so súvislosťou k, majúc n - procesorov, z ktorých t môže byť chybných. Potom problém byzantínskych generálov je riešiteľný bez podpisovania $\Leftrightarrow$ t < n/3 a t < k/2.





Veta 5 (Fischer, Lynch, Paterson, 1985) V plne asynchrónnom prostredí neexistuje 1-crash odolné riešenie problému dohody (konsenzu), dokonca, aj keď je vyžadovaná iba podmienka netriviálnosti.

Brach, Toueg (1985) - crash-odolný konsenzus je dosiahnuteľný s pravdepodobnosťou 1 ak t < n/2 (Byzantínsky-odolný ak t < n/3)

Dôkaz nemožnosti :

Lema 1 Neexistuje bivalentná inicializačná konfigurácia pre 1-crash odolný algoritmus dohody A

Ak A je netriviálny, existujú dosiahnuteľné konfigurácie 0- a 1-rozhodnutia; nech δ_0 a δ_1 sú počiatočné konfigurácie také, že konfigurácia v-rozhodnutia je dosiahnuteľná z δ_0.

7.7 Zložitosť

7.7.1 Horné odhady

Veta 6 (Fischer, Fowler, Lynch, 1982)
Nech $t < n/3$. Existuje t-odolné riešenie problému byzantínských generálov bez podpisovania, ktoré používa 2t+3 kôl výmeny informácie a $O(n.t + t^3 \log t)$ bitov správ.

Otvorený problém: Existuje algoritmus bez podpisovania, ktorý simultálne pracuje menej ako 2t+3 kôl a použije iba polynomiálny počet bitov ?

Veta 7 (Doler, Strong, Reischuk, 1982)

- Existuje t-odolné riešenie problému byzantínskych generálov s podpisovaním, ktoré používa t+1 kôl a posiela $O(n.t)$ správ.
- Existuje t-odolné riešenie problému byzantínskych generálov s podpisovaním, ktoré používa $O(t)$ kôl a posiela $O(n + t^2)$ správ.

- t+1 ohraničenie na počet kôl je príliš veľa pre praktické aplikácie.
- t+1 ohraničenie sa nedá vylepšiť v najhoršom prípade, keď sa skutočne vyskytne t chýb.

- Hľadajú sa prispôsobivé algoritmy, ktoré zastavia skôr ako sa vyskytne niekoľko chýb.

Algoritmus dosiahne

- *okamžitú dohodu*: ak všetky spoľahlivé procesory zastavia v tom istom kole
- *eventuálnu dohodu* inak

Veta 8 (Doler, Reischuk, Strong, 1982)
Nech t < n/3. Existuje t-odolný protokol, bez podpisovania, ktorý rieši problém byzantínskych generálov a dosahuje eventuálnu dohodu v minimálne $(2t+3, 2f+5)$ kolách, kde f $\leq$ t je skutočný počet chýb.

Veta 9 (Fischer, Lamport, 1985)
Existuje t-crash odolný protokol bez podpisovania, ktorý rieši problém generálov a dosahuje eventuálnu dohodu na konci kola f+2, kde f $\leq$ t je skutočný počet havárií (crashes).

algoritmus FL
{ správy - 0,1: generálove počiatočné hodnoty; ϕ: "Nepoznám"; NIL: implicitná hodnota; hodnota konsenzu sa vyberie z objavenej generálovej hodnoty, keď sa predchádza haváriam}

A. Kolo 1: Generál posiela svoju hodnotu každému procesoru

B. Kolo r, 1 < r $\leq$ t+1: Každý procesor robí:

1. Ak prijme od procesora hodnotu v $\in \{0, 1, NIL\}$ v kole r-1 tak zoberie hodnotu v ako svoju konsenzus hodnotu, pošle v každému procesoru a zastaví.
2. Inak, ak prijal od každého procesora ϕ počas kola r-1
tak zoberie NIL ako konsenzus; pošle NIL každému procesoru a zastaví.
3. Inak, pošle ϕ každému procesoru.

C. Koniec t+1 kola: Každý procesor, ktorý nezastavil robí:

1. Ak prijal v $\in \{0, 1, NIL\}$ počas t+1 kola tak zoberie v ako konsenzus hodnotu a zastaví.
2. Inak vyberie NIL ako hodnotu konsenzu a zastaví.

Korektnosť algoritmu FL - vyplýva zo skutočností:

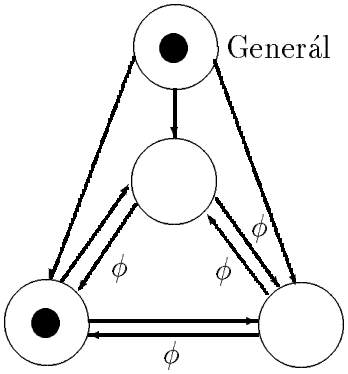
F1. Ak niektorý procesor zastaví v kroku B1 alebo B2 počas r-tého kola a vyberie hodnotu v, potom každý iný procesor, ktorý zastaví v kroku B1 alebo B2 počas kola r tiež vyberie hodnotu v.

F2. Ak niektorý procesor zastaví v B1 alebo B2 počas r-tého kola a vyberie hodnotu v, potom každý spoľahlivý procesor (ktorý ešte nezastavil) vyberie hodnotu w a zastaví v B1 v r+1 kole alebo v C1 v t+1 kole.

F3. Ak žiadny z procesorov nehavaruje alebo nezastaví počas r>1 kola, potom 0 je jediná správa poslaná počas kola.

F4. Ak niektorý procesor skončí v C2 v t+1 kole, potom všetky spoľahlivé procesory urobia to isté.

Naviac, ak menej ako k procesorov havaruje v prvých k-kolách, potom protokol skončí za k+1 kôl.



7.7.2 Dolné odhady

Veta 10 (Fischer, Lynch, 1982)

(a) Každý t-odolný protokol bez podpisov pre slabý problém konsenzu používa aspoň t+1 kôl výmeny správ v najhoršom prípade.

(b) Každý t-odolný autentifikovaný protokol pre problém byzantínskych generálov používa aspoň t+1 kôl výmeny správ v najhoršom prípade.

Slabý problém konsenzu s podpísanými správami - platí rovnaké ohraničenie ?

Veta 11 (Dolev, Strong, Reischuk, 1982)
Nech t $\leq$ n-2, a nech P je t-odolný protokol s podpisovaním riešiaci problém byzantínskych generálov, ktorý vždy dosiahne okamžitú dohodu . Potom je pre P možné bežať aspoň t+1 kôl aj keď sa nevyskytnú chyby.

Veta 12 (Dolev, Strong, Reischuk, 1982)
Nech P je t-odolný protokol s podpisovaním riešiaci problém byzantínskych generálov, ktorý dosiahne eventuálnu dohodu, a nech f < t. Potom je pre P možné bežať aspoň f+2 kôl iba s f chybami.

Môže byť tento výsledok rozšírený na t-crash odolné protokoly generálov ?

Malo by to ukázať optimalitu vety 9.

Veta 13 (Dolev, Reischuk, 1982)
Celkový počet správ a podpisov v akomkoľvek t-odolnom protokole byzantínskych generálov s podpisovaním je $\Omega(n.t)$.

Veta 14. (Dolev, Reischuk, 1982)
Celkový počet správ v akomkoľvek t-odolnom protokole byzantínskych generálov s podpisovaním je $\Omega(n + t^2)$.

Veta 7 ukazuje optimálnosť tohto ohraničenia pre autentifikované protokoly.