PLS(prefix labeling scheme) nad abecedou Σ je znackovanie vrcholov a hrán siete retazcami nad Σ^* tak, že

- vrcholy majú rôzne znacky
- v uzle majú všetky kanály rôzne znacky

PLS je *platná*, ak routovaním podla nej každá správa v konecnom case dosiahne svoj ciel.

packet pre d (prijatý/generovaný) v u

```
if d = l_u then doruc lokálne
else
    nech α_i je najdlhšie císlo kanála také, že α_i < d;
    send packet pre d po kanáli α_i
```

Veta 1 *Ku každému súvislému grafu G existuje platná PLS.*

Konštrukcia tree–PLS Nech T je kostra grafu G.

- koren je oznacený ϵ
- ak w je syn u tak $l_w = l_u.a_i$
  ak $u_1, \ldots, u_k$ sú rôzni synovia vrchola u, tak $a_1 \neq \ldots \neq a_k$ a $l_{u_i} = l_u.a_i$
- ak uw je frond, tak $\alpha_{uw} = l_w$
- ak w je syn u, tak $\alpha_{uw} = l_w$
- ak w je otec u, tak $\alpha_{uw} = \epsilon$ okrem prípadu, ked u má frond do korena; vtedy $\alpha_{uw} = l_w$

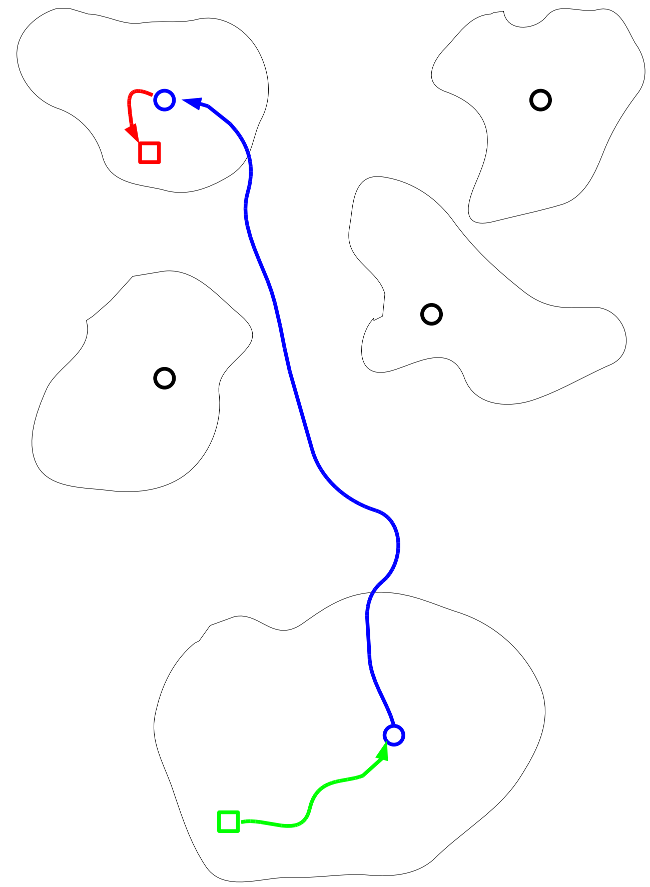
Fakt 1 *Pre popísanú tree–PLS platí*

1. *ak $u \neq v$ tak v u existuje kanál oznacený prefixom l_v*
2. *v situácii, ked u posiela správu do v cez w*
   (a) ak $u \in T[v]$, tak w je predchodca u, resp. v je predchodca w
   (b) ak u je predchodca v tak w je predchodca v bližšie k v (ako u)
   (c) ak $u \notin T[v]$, tak w je predchodca v alebo $d_T(w, v) < d_T(u, v)$

Dôsledok 1 *Ku $\forall$ grafu G s priemerom D_G $\exists$ PLS, ktorá $\forall$ packet doručí $\leq 2D_G$ hopov.*

Hierarchické routovanie

- siet rozdelená na disjunktné klastre
- centrum klastra
- komunikácia prostredníctvom centier klastrov
- packety môžu menit farby a spôsob routovania



spôsoby routovania

- v rámci klastra smerom ku centru – sink tree
- medzi centrami – kostra minimálnej velkosti
- v rámci klastra smerom k cielu – najkratšie cesty

$\forall s \in \mathbb{N}$ **existuje rozdelenie na klastre** $C_1, \ldots, C_m$ **tak, že**

- klastre sú **súvislé** podgrafy
- každý klaster obsahuje **aspon** s **vrcholov**
- každý klaster má **polomer nanajvýš** $2s$

konštrukcia

1. $D_1, \ldots, D_m$ disjunktné klastre velkosti aspon s s polomerom nanajvýš s
2. $\forall x \notin \bigcup D_i$ existuje najbližší $v_x \in \bigcup D_i;\; v_x \in D_{(x)}$
3. $d(x, x_v) \leq s$
4. $C_i = D_i \cup \{x \mid v_x \in D_i\}$

$\square$

Veta 2 *Ku každej sieti velkosti N existuje routovacia metóda, ktorá používa* ***tri farby*** *a* $\mathbf{O}(\sqrt{\mathbf{N}})$ ***rozhodnutí***

- $C_1, \ldots, C_m$ klastre velkosti aspon s, polomer $\leq 2s$; c_i centrum klastra C_i
- T- strom minimálnej velkosti obsahujúci všetky centrá;

  T má max. m listov

  T má max. $2m - 2$ vetviacich vrcholov (stupna > 2)

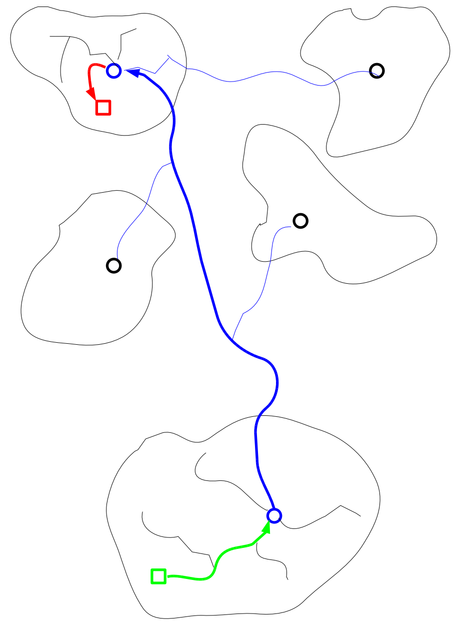
fázy routovania

1. zelená - žiadne rozhodnutie(sink tree)
2. modrá- max $2m - 2$ rozhodovaní v rámci stromu T
3. cervená - max $2s$ rozhodovaní v rámci cielového klastra

pocet rozhodnutí$\# \leq 2m - 2 + 2s \leq \mathbf{2N/s - 2 + 2s}$

$$\mathbf{s} \approx \sqrt{\mathbf{N}} \quad \Rightarrow \quad \# = \mathbf{O}\sqrt{\mathbf{N}}$$

$\square$



Veta 3 *Pre každú siet velkosti N a $f \leq \log N$ existuje routovacia metóda, ktorá používa* **$2f + 1$ *farieb*** *a* ***nanajvýš*** $O(f \cdot N^{1/f})$ ***rozhodnutí***

metóda – iteratívne na strom centier T

1. velkost klastrov
   - po prvej iterácii - N/s centier, N/s vetviacich vrcholov
   - po i iteráciách - m_i centier, m_i vetviacich vrcholov; potom po $i+1$ iteráciách - m_i/s centier, m_i/s vetviacich vrcholov

   $$m_f \leq N \left(\frac{2}{s}\right)^f$$

2. pocet farieb **iterácia pridáva 2 farby**

3. pocet rozhodovaní
   - $2m_f$ v rámci stromu "modré"
   - s v rámci jednej úrovne "cervené"

   $$\# \leq 2m_f + f \cdot s$$

   $$s \approx 2N^{1/f} \quad \rightsquigarrow \quad m_f = O(1), \# = O(f \cdot N^{1/f})$$

$\square$

Problém routovania packetov a veľkosť kanálov

M packetov, na začiatku umiestnených v N rôznych vrcholoch
$\mathbf{p_i}$ cieľ doručenia pre i-ty packet

one-to-one routing problem $\mathbf{p_i} \neq \mathbf{p_j}$ pre $1 \leq i < j \leq M$
dynamický model routing problému
packety generované náhodne v dostatočne dlhom časovom intervale

dvojrozmerná mriežka

greedy metóda (*routovanie po najkratších cestách*)
optimálny algoritmus na jednorozmernom poli
dvojrozmerná mriežka
$2\sqrt{N} - 2$ krokov **ale** neobmedzená veľkosť kanálov $O(\sqrt{N})$
random routing–(očakávaná) veľkosť kanála $O(1)$ s veľkou pravdepodobnosťou

pravdepodobnostné routovanie
čas $2\sqrt{N} + o(\sqrt{N})$, veľkosť kanála $O(\log N)$ s veľkou pravdepodobnosťou

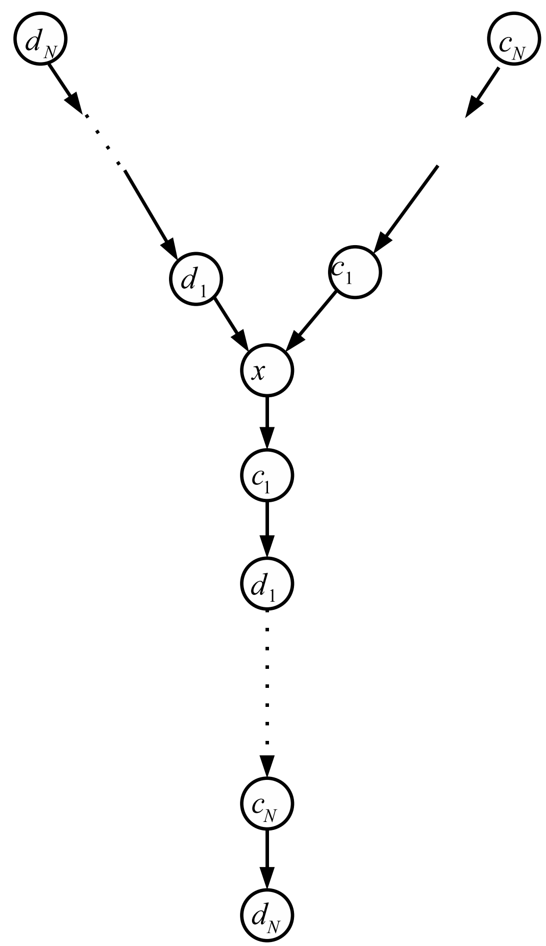
Routovanie na lineárnom poli $\Rightarrow$ keď sa packet potrebuje hýbať doľava resp. doprava, tak sa hýbe

- dobre definované, keď na začiatku má každý vrchol len jeden packet
- žiadne dva packety nechcú používať rovnakú hranu jedným smerom naraz
- každý packet je doručený v d krokoch, kde d je vzdialenosť cieľa

Základný greedy na dvojrozmernej mriežke

- neobmedzene veľké kanály/buffre
- prednosť majú packety, ktoré idú v príslušnom smere ďalej (farther-first)
- najskôr do správneho stĺpca, potom do správneho riadku

Vo všeobecnosti máme prípady, keď vyžadujeme veľkosť kanála $O(N)$



mriežka veľkosti $\sqrt{N} \times \sqrt{N}$, základný greedy algoritmus

1. analýza prvej fázy — pohyb v riadku $\rightsquigarrow$ packet dosiahne správny stĺpec počas prvých $\sqrt{N}$-1 krokov (na riadkových hranách nie je zdržanie) ! veľké buffre
2. analýza druhej fázy — **pohyb v stĺpci**

Lema 1 *Nech v n procesorovom* ***lineárnom poli*** *každý vrchol obsahuje ľubovoľný počet packetov, pričom neexistujú dva rôzne packety s rovnakým cieľom. Ak uprednostňujeme packet idúci ďalej, potom základný greedy algoritmus skončí routovanie po $n-1$ krokoch.*

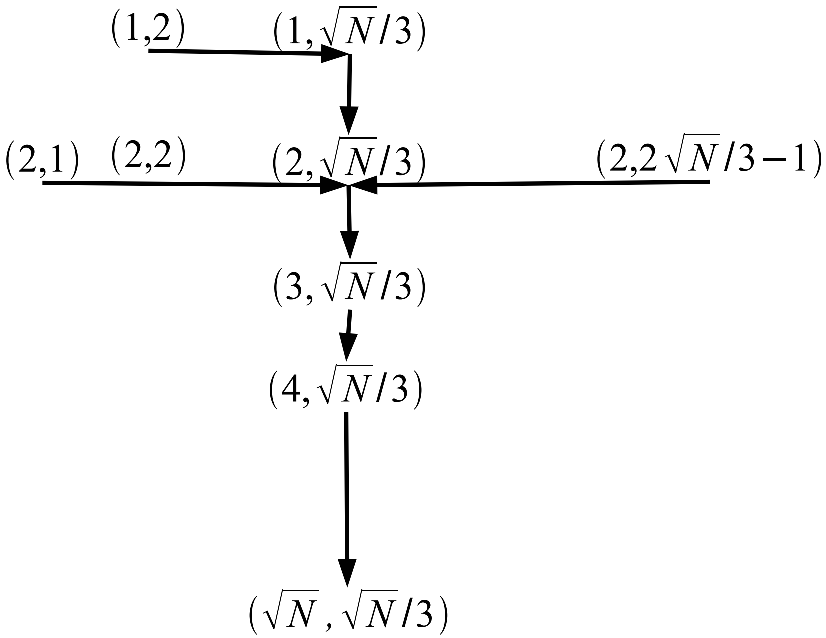
– packety smerujúce naľavo/napravo si navzájom neprekážajú (sledujeme smer doprava)
– MI: packet smerujúci do i najpravších vrcholov (prioritný) dosiahne jeden z i najpravších vrcholov počas $n-1$ krokov
  – najpravší nie je zdržiavaný nikým, druhý najpravší je zdržaný nanajvýš jedným, ...
  – i-ty najpravší je zdržiavaný nanajvýš $i-1$ krokov;
    ak je navyše najľavejší, potrebuje ešte $n-i$ krokov

$$i - 1 + (n - i) = n - 1 \qquad\qquad n = \sqrt{N}$$

čas: $\mathbf{2\sqrt{N} - 2}$ optimum
veľkosť kanála: veľká, až $\mathbf{\frac{2}{3}\sqrt{N} - 1}$

"najhorší prípad" pre veľkosť kanála

packety z vrcholov $(1,2), (1,3), \ldots, (1, \frac{\sqrt{N}}{3})$, a $(2,1), (2,2), \ldots, (2, \frac{2\sqrt{N}}{3} - 1)$
smerujú do vrcholov $(3, \frac{\sqrt{N}}{3}), (4, \frac{\sqrt{N}}{3}), \ldots, (\sqrt{N}, \frac{\sqrt{N}}{3})$,



- všetkých $\sqrt{N} - 2$ packetov dosiahne $(2, \frac{\sqrt{N}}{3})$ počas $\frac{\sqrt{N}}{3} - 1$ krokov, ale len $\frac{\sqrt{N}}{3} - 1$ packetov ho môže v tomto čase opustiť

- potrebujeme kanál veľkosti $\sqrt{N} - 2 - (\frac{\sqrt{N}}{3} - 1) = \frac{2}{3}\sqrt{N} - 1$

Ukážeme, že **v priemere** je to lepšie

- pre packety posielané do náhodných cieľov je *s pravdepodobnosťou blízkou 1* počet packetov v jednom kanále $O(1)$
- *očakávaný čas* zdržania packetu na ceste je - nezávisle od N a vzdialenosti - konštantý

Routovanie N packetov do náhodného cieľa

Priemerný prípad: na začiatku má $\forall$ vrchol packet s náhodne vygenerovaným cieľom $\Rightarrow$ v najhoršom prípade treba $N-1$ krokov (s pravdepodobnosťou $\frac{1}{N^{N-1}}$)

Fakt 2 *Maximálna veľkosť radu je* $O(\frac{\log N}{\log\log N})$ *s pravdepodobnosťou aspoň* $1 - O(1/N)$

- uvažujeme len stĺpce (na riadkoch sa nič nezdržiava)
- zdržanie nastáva, keď naraz prídu aspoň dva packety po riadku
- packet vo vrchole *zatáča*, ak prišiel po riadku a odchádza po stĺpci $\Rightarrow$ veľkosť radu odhadneme počtom packetov, ktoré vo vrchole zatáčajú
- na riadku zatáča nanajvýš $\sqrt{N}$ packetov

Pravdepodoobnosť, že aspoň r packetov zatočí v konkrétnom vrchole, je nanajvýš

$$\binom{\sqrt{\mathbf{N}}}{\mathbf{r}}\left(\frac{\mathbf{1}}{\sqrt{\mathbf{N}}}\right)^{\mathbf{r}}$$

Lema 2 *Pre každé* $0 < y < x$

$$\binom{x}{y} < \left(\frac{xe}{y}\right)^y$$

- $\binom{x}{y} \le \frac{x^y}{y!}$
- Stirling $z! = \frac{\sqrt{2\pi z}\cdot z^z}{e^z}(1 + \Theta(1/z))$

$$\sqrt{2\pi z}\left(\frac{z}{e}\right)^z \le z! \le \sqrt{2\pi z}\left(\frac{z}{e}\right)^{z+\frac{1}{12z}} \quad \Rightarrow z! > \left(\frac{z}{e}\right)^z, \text{ resp. } y! > \left(\frac{y}{e}\right)^y \qquad \square$$

Späť k pravdepodobnosti zatočenia vo vrchole $\binom{\sqrt{N}}{r}\left(\frac{1}{\sqrt{N}}\right)^r < \left(\frac{\sqrt{N}\cdot e}{r}\right)^r \left(\frac{1}{\sqrt{N}}\right)^r = \left(\frac{e}{r}\right)^r$

pre $r = \frac{e \log N}{\log\log N}$ máme

$$\left(\frac{\log\log N}{\log N}\right)^{\frac{e\log N}{\log\log N}} = 2^{-\frac{e\log N}{\log\log N}\cdot\log\left(\frac{\log N}{\log\log N}\right)} = \left(2^{-\log N}\right)^{\frac{e}{\log\log N}\cdot\log\left(\frac{\log N}{\log\log N}\right)} = o(N^{-2})$$

Neexistuje vrchol, v ktorom točí aspoň $r = \frac{e\log N}{\log\log N}$, s pravdepodobnosťou

$$1 - 4N \cdot o(N^{-2}) = 1 - o\left(\frac{1}{N}\right) \qquad \square$$

Tok packetov na stĺpcovej hrane

Zjednodušenie - tzv. **široký kanál**– kanálom ide toľko packetov, koľko treba

1. ohraničíme pravdepodobnosť toho, že počas Δ krokov ide aspoň $\frac{\alpha\Delta}{2}$ packetov konkrétnou stĺpcovou hranou v modeli so širokým kanálom
2. odstránime predpoklad o širokom kanáli

hrana **e** : $\quad \mathbf{(i,j) \longrightarrow (i+1,j)}$

- packet na hrane e v čase T musí byť na začiatku vo vzdialenosti T, pričom
- je v horných i riadkoch (max 2 na riadku vo vzd. T)
- smeruje do jedného z $\sqrt{N} - i$ vrcholov v stĺpci j

Pravdepodobnosť, že **to** nastane, je pre packety nezávislá, očakávaný počet je

$$\frac{2i\Delta(\sqrt{N} - i)}{N} \leq \frac{\Delta}{2}$$

očakávaný počet$\leq \frac{\Delta}{2}$ + (bez dôkazu) Lema 3

Lema 3 *Majme n nezávislých Bernouliho náhodných premenných $X_1, X_2, \ldots, X_n$, nech $Prob[X_k = 1] \leq P_k$ pre každé $1 \leq k \leq n$. Potom*

$$Prob[X \geq \beta P] \leq e^{(1-\frac{1}{\beta}-\ln\beta)\beta P}$$

$\beta > 1,\ X = X_1 + \cdots + X_n,\ P = P_1 + \cdots P_n$

Použitie:

$$n = 2i\Delta \qquad P_k = \frac{\sqrt{N}-i}{N} \quad \forall k \qquad P = \frac{2i(\sqrt{N}-i)\Delta}{N} \qquad \beta = \frac{\alpha N}{4i(\sqrt{N}-i)}$$

Pravdepodobnosť, že viac ako $\frac{\alpha\Delta}{2}$ packetov ide cez e počas Δ krokov je nanajvýš $e^{(\alpha-1\alpha\ln\alpha)\Delta/2}$ $//\beta \geq \alpha$, $\max i = \sqrt{N}/2$

Pre $\alpha = 1.5$ je pravdepodobnosť toho, že počas Δ krokov ide po konkrétnej hrane e viac ako $\frac{3\Delta}{4}$ packetov, nanajvýš $\mathbf{e^{-0.054\Delta}}$ $\square$

Lema 4 *Ak je packet p po kroku T vo vzdialenosti d od stĺpca s hranou e a ak v štandardnom modeli prechádza hranou e v kroku $T + d + \delta$, tak na hrane e musí byť nejaký packet v každom kroku intervalu $[T + d, T + d + \delta]$.*

Dôsledok 2 *Ak packet prechádza stĺpcovou hranou e v čase T modelu so širokým kanálom, a ak prechádza hranou e v kroku $T + \delta$ v štandarnom modeli, potom v každom kroku intervalu $[T, T + \delta]$ je na hrane e v štandardnom modeli nejaký packet.*

Lema 5 *Ak v štandardnom modeli počas Δ krokov $[T + 1, T + \Delta]$ prechádza cez e x packetov, potom existuje $t \geq 0$ také, že v modeli so širokým kanálom prechádza aspoň $x + t$ packetov po e počas $[T + 1 - t, T + \Delta]$*

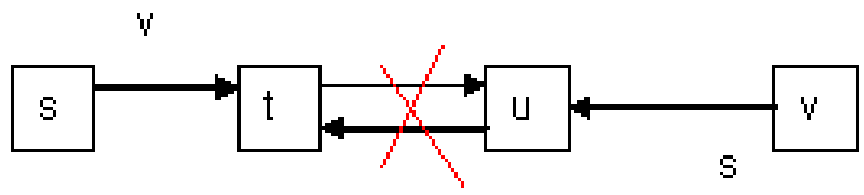
Lema 6 *Pravdepodobnosť, že pri použití základného greedy algoritmu prejde po konkrétnej stĺpcovej hrane e v priebehu Δ krokov aspoň $\alpha\Delta/2$ packetov je nanajvýš $O(e^{(\alpha-1-\alpha\ln\alpha)\Delta/2})$, $1 \leq \alpha \leq 2$.*

Dôsledok 3 *Pri použití greedy algoritmu platí, že s pravdepodobnosťou* $1 - O(1/N)$ *po žiadnej stĺpcovej hrane nejde súvislý úsek* $clogN$ *packetov, kde* $c = \frac{5\ln 2}{2\ln 2 - 1} < 9$

Veta 4 ***Pri použití základného greedy algoritmu na routovanie*** N ***packetov do náhodných cieľov v mriežke*** $\sqrt{N} \times \sqrt{N}$ ***je maximálny počet počet packetov na jedne hrane*** ≤ 4 ***s pravdepodobnosťou*** $1 - O(\frac{\log^4 N}{\sqrt{N}})$***. Navyše, pravdepodobnosť, že packet bude zdržaný o*** Δ ***je nanajvýš*** $O(e^{-\Delta}/6)$***.***

Prevencia uviaznutia

Každý vrchol má kanál veľkosti B (B buffrov, môže prijať B packetov).



s poslalo B packetov s adresou v do t

v poslalo B packetov s adresou s do u

} store-and-forward uviaznutie

// V tejto situácii ani u ani v nemôžu posielať packety ďalej.

Predpoklady

- Systém je graf $G = (V, E)$
- Každý kanál je modelovaný B bufframi

Spracovanie packetov vo vrchole u

- **generovanie** - vznik packetu
- **posúvanie** - presun packetu z buffra do buffra
- **spracovanie** (consuming) - odstránenie packetu z buffra

Kontroler

- spracovanie packetu v mieste doručenia je vždy povolené
- generovanie packetu vo vrchole so všetkými bufframi prázdnymi je povolené
- kontroler využíva len **lokálnu** informáciu

$\mathbf{Z_u}$ označuje množinu stavov v u, **M** množinu správ.

Kontroler je množina dvojíc kon$= \{Gen_u, For_u\}$, kde $Gen_u \subseteq Z_u \times M, For_u \subseteq Z_u \times M$.

Ak $c_u \in Z_u$ je stav(u), keď všetky buffre v u sú prázdne, tak $\forall p \in M (c_u, p) \in Gen_u$.

- Hovoríme, že packet je uviaznutý, ak podľa *kon* neexistuje postupnosť krokov, ktorá by viedla k jeho spracovaniu.
- Konfiguráciu voláme uviaznutie, ak obsahuje uviaznutý packet.
- Kontroler je bez uviaznutia, ak je uviaznutie z počiatočnej konfigurácie nedosiahnuteľné.

Štrukturované riešenie

G je sieť, **B** množina buffrov, $\mathcal{P}$ systém ciest v G odpovedajúci routovacím tabuľkám.

BG (buffer graph) (pre G,B) je orientovaný graf $BG = (B, BE)$ taký, že

- BG je **acyklický**
- $bc \in BE$ ak b, c sú buffre v tom istom vrchole, alebo sú to buffre v rôznych vrcholoch, ktoré sú spojené kanálom
- pre každú cestu $P \in \mathcal{P}$ existuje cesta v BG, ktorej **obrazom** je P.

  Nech $b_0, b_1, \ldots, b_s$ je cesta v BG, u_i vrchol, ktorému patrí buffer b_i. Potom pre postupnosť $u_0, u_1, \ldots, u_s$ platí $u_i u_{i+1} \in E$, alebo $u_i = u_{i+1}$. Ak z tejto postupnosti vynecháme súvislé duplikáty (inými slovami spravíme najdlhšiu podpostupnosť $u_{j1}, u_{j2}, \ldots, u_{jk}$ takú, že $u_t \neq u_{t+1}$ pre každé $t = j1, \ldots, j(k-1)$), dostaneme obraz cesty P.

p je packet vo vrchole u s adresou v; hovoríme, že buffer b je vhodný pre p, ak v BG existuje cesta z bufra b do buffra c vo vrchole v, ktorej obrazom je cesta v G, po ktorej sa p môže dostať z u do v.

Jedna z nich je **garantovaná** a **nb(p, b)** označuje prvý buffer na nej (nb = next buffer).
fb(p) – buffer v u vhodný pre packet p novovygenerovaný vo vrchole u (fb = first buffer).

Buffer-graf kontroler

bgc$_{\mathbf{BG}}$ kontroler:

- generovanie je povolené $\Leftrightarrow$ $fb(p)$ je voľný;
  vygenerovaný packet je umiestnený do tohto buffra
- forwardovanie z buffra v u do buffra vo w (možno u=w) je povolené $\Leftrightarrow$ $nb(p,b)$ vo w je voľný;
  ak sa forwarduje, p sa presunie do $nb(p,b)$

Fakt 3 *bgc_{BG} je kontroler bez uviaznutia*

rád buffra b – dĺžka najdlhšej cesty v BG, ktorá končí v b

R – maximálny rád

IP – $\forall s$ $r < s \leq R$ žiaden buffer rádu s neobsahuje uviaznutý packet

Destination scheme

V každom vrchole u je buffer $\mathbf{b_u[v]}$ pre každú destináciu v. Potom stačí definovať BG nasledovne:

$\mathbf{BG_d = (B, BE)}$

$$(b_u[v1]b_w[v2]) \in BE \;\Leftrightarrow\; v1 = v2 \text{ a } uw \text{ je hrana } T_{v1} \text{ (sink tree s kreňom v1).}$$

Ak packet p generovaný v u má adresu v, tak $fb(p) = b_u[v]$. Ak ho posúvame do w, tak $nb(p, b) = b_w[v]$

Ak takto spravíme kontroler, bude bez uviaznutia. Chce to veľa buffrov.

Hop-so-far scheme

k je **dĺžka najdlhšej cesty** v grafe, v každom vrchole je $\mathbf{k+1}$ **buffrov**(pre každú potenciálnu dĺžku jeden); správa si počíta, koľko hopov už spravila.

$\mathbf{BG_h = (B, BE)}$

$$(b_u[i]b_w[j]) \in BE \;\Leftrightarrow\; i+1=j \;\text{ a } uw \in E$$

K ceste $P = u_0, u_1, \dots, u_l$ v grafe G je $b_{u0}[0], b_{u1}[1], \dots, b_{ul}[l]$ postupnosť buffrov v BG_h.

Definujme

$\mathbf{fb(p) = b_u[0]}$ pre p generované v u,
$\mathbf{nb(p, b_u[i]) = b_w[i+1]}$ pre packet, ktorý sa posúva z u do w.

Fakt 4 *Pre ľubovoľný súvislý graf je takto zadefinovaný kontroler bez uviaznutia, ak sa správy routujú po min-hop cestách.*

Orientovania grafu

Acyklická orienácia grafu G je orientovaný acyklický graf, ktorý vznikne z G zorientovaním hrán. Postupnosť $G_1, ..., G_B$ acyklických orientácií grafu G je acyklické pokrytie veľkosti **B** pre množinu ciest $\mathcal{P}$, ak $\forall P \in \mathcal{P}$, P možno napísať ako zreťazenie nanajvýš B ciest $P_1, \ldots, P_k$, $k \leq B$ takých, že P_i je cesta v G_i.

Ak vieme spraviť acyklické pokrytie veľkosti B, stačí nám vo vrchole B buffrov.
Pri **generovaní** packetu p vo vrchole u sa tento umiestni do buffra $b_u[1]$ $\mathbf{fb(p) = b_u[1]}$
Pri **forwardovaní/posúvaní** packetu z vrchola u do vrchola w číslo buffra ostáva, ak môžeme pokračovať v grafe G_i; inak preskočíme do grafu G_{i+1}.

$\mathbf{BG_a = (B, BE)}$,kde

$$(b_u[i]b_w[j]) \in BE \Leftrightarrow (uv \in E) \wedge (i = j \wedge uw \in E_i) \vee (i + 1 = j \wedge wu \in E_i)$$

$$\mathbf{fb(p)} = b_u[1] \qquad \mathbf{nb(p, b_u[i])} = \begin{cases} b_w[i], & \text{ak } uw \in E_i; \\ b_w[i+1], & \text{ak } uw \notin E_i. \end{cases}$$

Fakt 5 *Kontroler $acoc = bgc_{BG_a}$ je bez uviaznutia.*

kruh

Fakt 6 *Pri použití kontrolera acoc v kruhu stačia tri buffre vo vrchole.*

Nech $d_c(u,v)$ je vzdialenosť v smere hodinových ručičiek
$d_a(u,v)$ označuje vzdialenosť proti smeru hodinových ručičiek
C označuje súčet váh
$d(u,v) = \min\{d_c(u,v), d_a(u,v)\}$

I. Ak existujú vrcholy u, v také, že ležia vo vzdialenosti $C/2$, zoberiem dva vrcholy u, v vo vzdialenosti $C/2$ a zorientujeme cesty

$$G1 = G3 \quad \text{od } u \text{ do } v \qquad\qquad G2 \text{ od } v \text{ k } u$$

II. Neexistujú vrcholy u, v také, že ležia vo vzdialenosti $C/2$ - vtedy zoberme ako u, v také vrcholy, že $d(u,v)$ je najbližšie k $C/2$

strom

Fakt 7 *Pri použití kontrolera acoc v strome stačia dva buffre vo vrchole.*

vrchol v, sink-tree T_v — od koreňa k listom, resp. od listov ku koreňu.

Neštrukturované riešenie (Toueg, Ullman 81)

forward-count - **FC**: packet p si so sebou nesie informáciu $\mathbf{s_p}$ o tom, koľko hopov ešte chýba do cieľa; $\mathbf{f_u}$ je počet voľných buffrov v u.
Packet sa akceptuje $\Leftrightarrow$ $s_p < f_u$

backward-count - **BC**: packet p si so sebou nesie informáciu $\mathbf{t_p}$ o tom, koľko hopov už spravil; $\mathbf{f_u}$ je počet voľných buffrov v u. Nech k je maximálna dĺžka cesty v grafe G.
Packet sa akceptuje $\Leftrightarrow t_p > k - f_u$

nevýhoda – vo vrchole musí byť aspoň taký po4et buffrov, ako dĺžka najdlhšej najkratšej cesty

Fakt 8 *Ak $B > k$, FC-kontroler je deadlock-free.*

Dôkaz sporom - nech γ **je dosiahnuteľné uviaznutie**. Uvažujme konfiguráciu δ takú, ktorá sa dá dosiahnuť **z** γ **maximálnym počtom krokov** - forward, spracovanie.

Keďže v δ sa žiaden packet nemôže posunúť a γ je uviaznutie, v δ je aspoň jeden packet. Nech p je ten z nich, ktorý má minimálnu vzdialenosť do cieľa, u vrchol, v ktorom sa nachádza.

Keďže u nie je jeho cieľom, existuje sused w vrchola u, do ktorého má byť p posunuté. Keďže sa žiaden krok nedá realizovať, $\mathbf{s_p - 1 \geq f_w}$. Súčasne $s_p \leq k < B$, takže $f_w < B$ a vo w musí byť (v konfigurácii γ) aspoň jeden packet.

Nech q je ten packet, ktorý bol akceptovaný **posledný**, f'_w je počet voľných buffrov tesne predtým, ako bolo q akceptované.
Zrejme $f'_w \leq f_w + 1$ (možno sa niektorý buffer medzičasom uvoľnil)

$$s_q < f'_w \leq f_w + 1 \leq s_p$$

To je spor s voľbou p.

$\square$

Vo vrchole si budeme pamätať vektor $(j_0, \ldots, j_k)$, udávajúci v i-tej položke počet packetov so vzdialenosťou od cieľa rovnou i.

Forward State - **FS**: packet p je vo vrchole u akceptovaný práve vtedy, keď

$$\forall i, 0 \leq i \leq s_p : i < B - \sum_{s=i}^{k} j_s$$

Backward State - **BS**: packet p je vo vrchole u akceptovaný práve vtedy, keď

$$\forall j, t_p \leq j \leq k : j > \sum_{t=0}^{j} i_t - B + k$$

Fakt 9 *Každý krok, akceptovaný FC kontrolerom je povolený aj FS kontrolerom.*

$$s_p < f_u = B - \sum_{s=0}^{k} j_s \Rightarrow i \leq s_p; i < B - \sum_{s=i}^{k} j_s$$

Prehľadávanie grafov

- Fixovaná topológia
- Neorientovaný graf
- Súvislý graf
- Asynchrónna komunikácia

decide/**rozhodni** – špeciálna int. udalosť; **cieľ** vlnového alg. – dosiahnuť rozhodnutie

Vlnový algoritmus je taký distribuovaný algoritmus, ktorý spĺňa

1. **terminácia** - $\forall C : |C| < \infty$
2. **rozhodnutie** - $\forall C\ \exists e \in C$: e je decide udalosť
3. **závislosť** – v každom výpočte každej decide udalosti/rozhodnutiu predchádza udalosť v každom procese $\forall C\ \exists e \in C\ (e \text{ je decide} \Rightarrow \forall q \in P\ \exists f \in C_q :\ f \leq e)$

Výpočtu vlnového algoritmu hovoríme **vlna**.

Traverzovací algoritmus/traverzovanieje vlnový algoritmu, ktorý navyše spĺňa

- algoritmus má jediného iniciátora, ktorý jediný rozhoduje
- existuje úplné usporiadanie procesov

Vlnové algoritmy – vlastnosti

Fakt 10 *Ku každej udalosti e existuje iniciátor p a udalosť f v C_p, ktorá jej predchádza.*

Fakt 11 *Nech C je vlna s jediným iniciátorom p, $father_q$ je proces, od ktorého neiniciátor q dostal prvú správu. Potom*

$T = (P, E_T)$, *kde* $E_T = (qr), q \neq p \wedge r = father_q$ *je kostra.*

Fakt 12 *Nech C je vlna, d_p decide udalosť v p. Potom*

$\forall q \neq p \exists f \in C_q : (f \leq d_p \wedge f$ *je send udalosť*$)$

Veta 5 *Nech C je vlna s jediným iniciátorom p, rozhodnutie d_p sa robí v p. Potom v C sa vymení* ***aspoň*** *N* ***správ****.*

Veta 6 *Nech A je vlnový algoritmus pre ľubovoľnú sieť bez znalosti mien susedov. Potom v každom výpočte A sa vymení* ***aspoň*** $|E|$ ***správ***

PIF(propagation of information with feedback):

$M-$ množina procesov, ktoré majú (rovnakú) správu. Úlohou je, aby všetky ostatné procesy túto správu dostali a akceptovali.

Jeden musí dostať potvrdenie, že ju všetci dostali

Fakt 13 *Každý PIF môže byť implementovaný ako vlna a naopak.*

SYN - synchronizácia je charakterizovaný

- v každom procese q sa musí realizovať udalosť a_q
- v niektorých procesoch sa má realizovať b_p, ktorej musia predchádzať všetky a_q
- konečne veľa vymenených správ
- b_p budeme považovať za decide udalosť

Fakt 14 *Každý algoritmus na riešenie SYN je vlna, každú vlnu možno realizovať ako SYN algoritmus.*

Počítanie infima - INF.

- $(X, \leq)$ je množina s čiastočným usporiadaním
- c je **infimum** a, b ak $c \leq a, c \leq b \; \forall d((d \leq a \wedge d \leq b) \Rightarrow (d \leq c))$
- X je taká, že infimum vždy existuje.

Fakt 15 *Každý INF je vlna; každú vlnu možno použiť na počítanie INF.*

Vlna na kruhu (resp. grafe s HK)

iniciátor:
send (tok) do $next_p$; receive(tok); decide

neiniciátor:
receive(tok); send (tok) do $next_p$

Veta 7 *Algoritmus vlna na kruhu je vlna.*

Vlna na strome s inicátormi v listoch

Iniciátori –listy

Neiniciátor
- keď vrchol príjme správu od všetkých susedov až na jedného, pošle mu správu
- po prijatí správy od každého suseda vrchol rozhodne

$\mathbf{rec_p[q]}$ – či dostalo p správu od q; inicializovaná na false
$\mathbf{Neigh_p}$ – množinu susedov vrchola p

while $\sharp\{q : rec_p[q] = false\} > 1$ **do**
    receive (tok) od q
    $rec_p[q] \leftarrow true$
send (tok) do q_0, pre ktoré $rec_p[q_0] = false$ ▷ ostal len jeden vrchol - otec
receive (tok) od q_0; $rec_p[q_0] \leftarrow true$; decide
for $q \in Neigh_p, q \neq q_0$ **do**
    send (tok) do q.

Veta 8 *Algoritmus Vlna na strome s iniciátormi v listoch je vlna.*

konečnosť - každý posiela nanajvýš jednu správu, preto sa dosiahne terminálna konfigurácia.

V tejto **terminálnej** konfigurácii γ nastane aspoň v jednom procese udalosť decide

V každom procese predchádzalo decide nejaké send.

Shout & Echo- **centralizovaný algoritmus**, verzia prehľadávania do šírky

$\mathbf{rec_p}$ počíta prijaté správy
$\mathbf{father_p}$ určuje otca vo vznikajúcej kostre

Iniciátor:

for $q \in Neigh_p$ **do** send (tok) do q

while $rec_p < |Neigh_p|$ **do**
receive(tok)
$rec_p \leftarrow rec_p + 1$

decide

Neiniciátor:

receive (tok) od q
$father_p \leftarrow q; rec_p \leftarrow rec_p + 1$
for $q \in Neigh_p q \neq father_p$ **do** send (tok) do q

while $rec_p < |Neigh_p|$ **do**
receive(tok); $rec_p \leftarrow rec_p + 1$

send (tok) do $father_p$.

Veta 9 *Algoritmus Shout & Echo je vlna. Pri jeho realizácii sa poslalo $2|E|$ správ, časová zložitosť je $2D$.*

Fázový algoritmus - centralizovaný vlnový pre orientované grafy
predpokladá **znalosť priemeru** D (alebo horný odhad D', napr. $N-1$). Po hrane sa posiela presne D správ susedovi. Po prijatí i správ od $\forall$ suseda sa spúšťa fáza pre $i+1$.

In_p je množina vchádzajúcich hrán; Rec_p slúži na počítanie prijatých správ;
Out_p je množina odchádzajúcich hrán; $Sent_p$ počíta poslané správy

if p je iniciátor **then**
    for $r \in Out_p$ **do**
        send(tok) do r;
    inc($Sent_p$);
while $min_q Rec_p[q] < D$ **do**
    receive (tok) od q_0; inc($Rec_p[q_0]$)
    if $min_q Rec_p[q] \geq Sent_p \wedge Sent_p < D$ **then**
        for $r \in Out_p$ **do**
            send (tok) do r;
        inc($Sent_p$);
decide.

Počet správ - po každej hrane posielame D-krát susedovi, D-krát od neho $\rightarrow$ $\mathbf{2|E|D}$.

Finn - decentralizovaný vlnový algoritmus– orientovaný silne súvislý graf s jednoznačnou identifikáciou procesov.

$\mathbf{Inc_p}$ množina procesov q, v ktorých sa vykonala udalosť pred poslednou udalosťou v p

$\mathbf{NInc_p}$ množina procesov q, ktorých všetci susedia r realizovali nejakú udalosť pred poslednou udalosťou v p

```
if p je iniciátor then
    for r ∈ Out_p do
        send [Inc_p, NInc_p] do r
while Inc_p ≠ NInc_p do
    receive[Inc_s, NInc_s] od s; rec_p[s] ← true;
    Inc_p ← Inc_p ∪ Inc_s; NInc_p ← NInc_p ∪ NInc_s;
    if ∀q ∈ In_p : rec_p[q] then NInc_p ← NInc_p ∪ {p}
    if Inc_p alebo NInc_p sa zmenilo then
        for r ∈ Out_p do
            send [Inc_p, NInc_p] do r
decide.
```

\# správ celkovo – **2N|E|**

Traverzovanie

1. Jeden iniciátor, ktorý štartuje poslaním jednej správy
2. Proces po prijatí správy alebo pošle správu alebo rozhodne
3. Algoritmus terminuje v jedinom procese; v tom čase už každý proces poslal správu

T1: Proces môže vykonať ľubovoľný konečný počet udalostí v nulovom čase (lokálne výpočty sa zanedbávajú)

T2: čas medzi send a receive je nanajvýš jednotkový

čas traverzovacích algoritmov sa rovná počtu správ

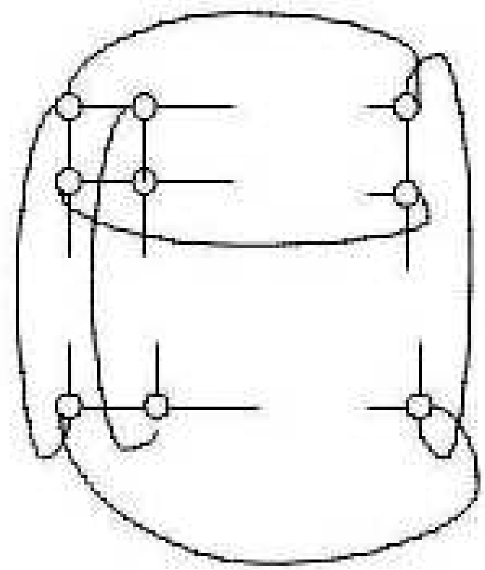
Definícia 1 *Traverzovací algoritmus je* $f(x)-$traverzovací, *ak po vykonaní x krokov (posunov tokena), je navštívených $max f(x), x$ vrcholov*

Fakt 16 *Traverzovanie torusu a hyperkocky sú $x-$traverzovacie algoritmy.*

Traverzovanie torusu

Torus je "zacyklená mriežka"
$G = (V, E)$, kde $V = Z_n \times Z_n = \{(i,j), 0 \leq i, j < n\}$
$E = \{((i,j)(I,J)); (I = i \wedge J = j \pm 1) \vee (I = i \pm 1 \wedge J = j)\}$

Predpokladáme **zmysel pre orienáciu** - vrcholy vedia, kde je hore-dole-vpravo-vľavo



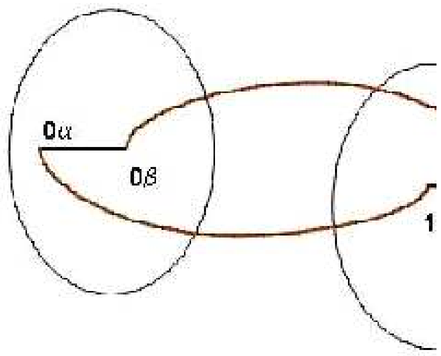
iniciátor
send (num, 1) **Up**;

po prijatí $(\mathbf{num}, \mathbf{k})$
if $k = n^2$ **then** decide
else if $n|k$ **then** send $(num, k+1)$ **Up**
else send $(num, k+1)$ **Right**

traverzovanie hyperkocky

Hyperkocka je graf $G = (V, E)$, kde

V $= \{(b_0, b_1, \ldots, b_{n-1})\}$
E $= \{((b_0, b_1, \ldots, b_{n-1})(c_0, c_1, \ldots, c_{n-1}));$
b, c sa líšia v jedinom bite$\}$



iniciátor

send $(num, 1)$ **kanálom n-1**

po prijatí (num, k)

if $k = 2^n$ **then** decide
else
nech d je maximálne také, že $2^d | k$;
send $(num, k + 1)$ **kanálom d**

Traverzovanie súvislých sietí - Tarryho prieskum

R1. Nikdy nie dvakrát po tom istom kanáli

R2. Neiniciátor posiela otcovi až vtedy, keď nie je iná možnosť

Garantuje to

- Každý kanál susediaci s iniciátorom prejdeme dvakrát
- Neiniciátor-každý susediaci kanál prejdeme raz v každom smere
- Každý proces bol navštívený

T[p] – vrcholy v podstrome s koreňom **A[p]** - vrcholy na ceste z koreňa do p

Kostra je **DFS kostra**, ak pre každý frond pq platí: $q \in T[p]$ alebo $q \in A[p]$

Klasický DFS = Tarry +

R3: Prijatý token vraciame po tom istom kanáli ak to pravidlá R1, R2 umožňujú

Distribuovaný DFS- Cheung´83 pre p

Iniciátor/iniciácia

$father_p \leftarrow p$; zvoľ $q \in Neigh_p$;
send(tok) do q

Father$_\mathbf{p}$ - od koho prišiel prvý token

Neigh$_\mathbf{p}$ - susedia

Used$_\mathbf{p}$ - už sme tadiaľ posielali

Receive(tok) od s

if $father_p$ undef **then** $father_p \leftarrow s$;
if $\forall q \in Neigh_p$: $used_p[q]$ **then** decide
else
  if $\exists q \in Neigh_p : (q \neq father_p \wedge \neg used_p[q])$ **then**
    if $(father_p \neq s \wedge \neg used_p[s])$ **then** $q \leftarrow s$
    else zvoľ $q \in Neigh_p \setminus \{father_p\}$ a $\neg used_p[q]$
      $used_p[q] \leftarrow$ true; send(tok) do q;
  else $used_p[father_p] \leftarrow$ true; send(tok) do $father_p$.

Počet správ $2|E|$
Čas $2|E|$ jednotiek

Awerbuch 85– DFS v lineárnom čase

Zefektívnenie – **navštíviť každý** vrchol, **posielať** len **po stromových** hranách.

Metóda - navštívený uzol pošle ***visited*** susedom okrem otca; po získaní všetkých potvrdení ***ack*** pošle ***discover*** nenavštívenému

Globálna informácia o už navštívených vrcholoch sa posielaním správ visited udržiava v ich susedoch. Vrchol poslaním ack potvrdzuje prijatie visited.

discover - presun z navštíveného do nenavštíveného; posielanie tok

visited - informovanie susedov ack - potvrdenie prijatia správy

return - presun z aktuálneho vrchola k otcovi

Neigh$_{\mathbf{b}}$ -susedia

father$_{\mathbf{p}}$ -otec; inic. $Father_p$:=p

unvisited - od ktorých ešte nebolo prijaté visited

flag - 1 v čase medzi poslaním visited a prijatím ack; inic flag(p,q):=0

Inicializácia– sám sebe pošle discover

Awerbuch 85– spracovanie správy vo vrchole p

discover od s

$father_p$:=s;
forall $q \in Neigh_p \setminus \{s\}$ do **begin**
    send visited do q; $flag(p,q)$:=1 **end**
if $Neigh_p = \{q\}$ **then** return do s

return od s

if $\exists q \in unvisited$ **then begin** send discover do q; unvisited:= unvisited$\setminus\{q\}$ **end**
else if $father_p \neq p$ **then** return do $father_p$ **else** decide

visited od s

unvisited:= unvisited$\setminus\{s\}$;
send ack do s

ack od s

flag(p,s):=0;
if flag(p,s):=0 $\forall q \in Neigh_p$ **then** deliver return sám sebe
Počet správ - 4|E| **Čas - 4N-2**

Lakshmanan, Meenakshi, Thulasiraman - vylepšený Awerbuch

posielame správy - discover, visited, return

čas sa šetrí **neposielaním** ack. Keď treba, recover z chyby

visited od s

nomessage:= nomessage $\setminus\{s\}$; recover

discover od s

nomessage:= nomessage$\setminus\{s\}$;
recover;
if not visited$\{p\}$ **then begin**
  visited$\{p\}$:=true; $father_p = s$;
  presuň aktivitu;
  forall $q \in Neigh_p \setminus \{father_p, explore(p)\}$ **do** send visited do q
end

return od s
nomessage:= nomessage $\setminus\{s\}$; presuň aktivitu

recover inicializuj zotavenie z chyby, ak treba
if explore(p)=s **then** presuň aktivitu

presuň aktivitu

if $\exists q \in nomessage$ **then begin**
  explore(p):=q; send discover do q
end
else begin
  explore(p):=p;
  if $father_p = p$ **then** decide
  else send return do $father_p$
end

Čas = 2N-2

DFS pri znalosti susedov

Zakomponovaním visited vrcholov do tokenu sa môžeme vyhnúť posielaniu cez frond $\Longrightarrow$ **2N-2správ**, **čas** tiež **2N-2**

Počítanie súčtu

vlnový algoritmus vo všeobecnosti nemôžme použiť na počítanie súčtu

traverzovací algoritmus do procesov vložíme j_p (hodnoty, ktoré chceme sčítať)
token nesie informáciu o súčte – pri navštívení vrchola p $s := s + j_p;\ j_p := 0$

využitím kostry proces p posiela otcovi súčet podstromu s koreňom p

využitím identifikačných čísel evidujeme si aj čísla procesorov, ktoré sme už sčítali – veľká bitová zložitosť

Prehľadávanie grafov

1. shout and echo
2. do šírky
   - **algoritmus Cheung'83** s kubickou komunikáciou a lineárnym časom Každá správa obsahuje počítadlo hopov. Na začiatku iniciátor pošle všetkým svojim susedom správu 1. Každý vrchol má lokálnu premennú (inicializovanú na nekonečno), v ktorej si uchováva vzdialenosť od iniciátora. Keď dostane správu i a jeho doterajšia vzdialenosť je väčšia, nastaví si svoju vzdialenosť na i a pošle správu i+1 svojim susedom.
   - **algoritmusCheung,Zhu'87** s kvadratickou komunikáciou aj časom Podobne ako predchádzajúci, ale kostra sa vytvára po vrstvách. V každej fáze iniciátor pošle po už vybudovanej časti kostry správu. Aktuálne listy pošlú správu do vzdialenosti 1, čím vyrobia novú vrstvu. Kvôli synchronizácii sa pošlú potvrdenia k iniciátorovi.
3. do hĺbky
   - **traverzovací algoritmus** so zložitosťou 2m
   - algoritmus **Awerbuch'85**
   - algoritmus **Cidon'88**