Distribuované algoritmy

Distribuovaný výpočet

každý procesor má vlastnú pamäť a komunikuje s ostatnými len pomocou správ (message passing)

v SPMD modeli má každý procesor kópiu spoločného (sekvenčného) kódu.

od ostatných sa líši

- jednoznačným identifikátorom (**myid**) ... rank
- informáciou o komunikačných (orientovaných) kanáloch, cez ktoré možno prenášať správy (podľa topológie)

Problémy distribuovaného systému

- procesor nemá informáciu o (aktuálnom) globálnom stave systému
- neexistuje globálny čas – udalosti nie sú globálne usporiadané podľa času ich vzniku
- vykonávanie procesov je nedeterministické (rozdiely v rýchlosti vykonávania)
- príjem správ nemusí byť v tom poradí, v akom boli odoslané
- výpočtové a komunikačné chyby

Synchrónny distribuovaný výpočet

Vykonávanie simultánnych krokov so serializovanými (FIFO) bezpečnými prenosmi správ

Krok synchrónneho výpočtu obsahuje:

- prijatie všetkých správ zo vstupných komunikačných kanálov (ak sa tam nejaké nachádzajú)
- sekvenčný výpočet (na základe prijatých správ), príprava ďalších správ a ich pripojenie ku príslušným komunikačným kanálom
- odoslanie správ do zvolených kanálov (ak je to potrebné)

Časová zložitosť – počet týchto krokov, potrebných na dosiahnutie koncového stavu

Komunikačná zložitosť – počet odoslaných správ (presnejšie - počet bitov v správach)

Základné distribuované funkcie

- **send** (data, dest, tag) synchrónny (blokovaný) resp. asynchrónny (neblokovaný)
- **receive** (data, src, tag, status)

dest, src - identifikácia procesu (komunikačného kanála), s ktorým chceme komunikovať (alebo ANY_SOURCE),
tag – značka správy (alebo ANY_TAG),
status – ukazovateľ na reálne prijaté hodnoty src a tag

skrátene **receive** (data, src, tag) s pemennými src resp. tag – ako ANY_TAG resp. ANY_SOURCE s priradením (src,tag)=status

- **broadcast** (data, src) odosielanie aj príjem
- **scatter** (data, datapart, src), **gather** (datapart, data, dst)
- **map**, **reduce**

Distribuovaný algoritmus

Korektnosť distribuovaného algoritmu:

- dôkaz správnosti výsledku
- dôkaz ukončenia všetkých procesov
- dôkaz vyprázdnenia všetkých komunikačných kanálov

Leader election

Voľba koordinátora (leadera) v kruhovej sieti (ring)

- každý procesor je spojený v kruhu s ľavým a pravým susedom
- výsledkom je dosiahnutie špeciálneho stavu *leader* pre jediný procesor
- ak sú všetky procesory identické, problém výberu koordinátora nie je možné riešiť
- každý proces má jednoznačný identifikátor **myid,** vyhráva ten, ktorého identifikátor je najvyšší
- riešenie porušením symetrie (symmetry-breaking)

Leader election

LCR algoritmus (LeLann, Chang, Roberts) pre jednosmerný kruh

- každý proces pošle svoj identifikátor do kruhu
- z opačnej strany prijíma id, ktoré porovná so svojím, ak je vyššie – prepošle po kruhu ďalej
- koordinátorom je ten, kto dostane svoje id po obehnutí celého kruhu

- časová zložitosť – $O(n)$
- komunikačná zložitosť – $O(n^2)$ ... priemerne $O(n \cdot \log n)$

Ring leader election (LCR)

```
procedure RingLeaderElection
{vstup: in, out – rozhrania, výstup: status – leader alebo not leader}
status := not leader; send(myid, out, 0); receive(i, in, tag);
while tag = 0 do
    if i > myid then send(i, out, 0)
                else if i = myid then send(i, out, 1); status := leader endif
    endif
    receive(i, in, tag)
endwhile
if status <> leader then send(i, out, 1)
endif
end RingLeaderElection
```

Bidirectional ring leader election

HS (Hirschberg, Sinclair) algoritmus pre obojsmerný kruh

- v k-tej fáze posiela proces svoje id len do vzdialenosti 2^k do oboch strán, správu zruší proces s vyšším číslom
- ak sa mu vráti id z opačnej strany – je víťazom

v k-tej fáze začína najviac $\left\lfloor \frac{n}{2^{k-1}+1} \right\rfloor$ procesov

v k-tej fáze každý proces spôsobí najviac 2^{k+2} správ, spolu ich teda v k-tej fáze bude najviac 4n a vykoná sa najviac 2^{k+1} krokov

fáz je najviac log n, teda celkovo O(n . log n) správ a najviac $2^{\log n+3} = 8n$ krokov –> čas výpočtu bude v O(n)

Bidirectional ring leader election (HS)

procedure HSRingLeaderElection

{vstup myid > 0, in, out; výstup: status – leader alebo not leader}

status := **not** leader; decide:=false; phase:=0; echo:=0; **send**(**myid**,in,1); **send**(**myid**,out,1);

while not decide **do receive**(i, dir, tag); {dir – direction in = 0; out = 1}

if i > **myid then**
if tag > 1 **then send**(i, 1-dir, tag-1) {1-dir je opačné rozhranie k dir}
else if tag = 1 **then send**(i, dir, 0) **else send**(i, 1-dir, 0) **endif endif**

else if i = **myid then if** echo = 0 **then** echo:=1 {očakáva potvrdenie z oboch strán}
else if tag > 0 **then** status := leader; **send**(0, out, **myid**) {terminácia}
else phase := phase+1; **send**(**myid**, in, 2^{phase}); **send**(**myid**, out, 2^{phase});
echo := 0 **endif endif**

else if i = 0 **then** decide := true; **if** status <> leader **then send**(0, out, tag) **endif endif**

endif endif

endwhile

end HSRingLeaderElection {posielanie správ send je neblokované ... Bsend}

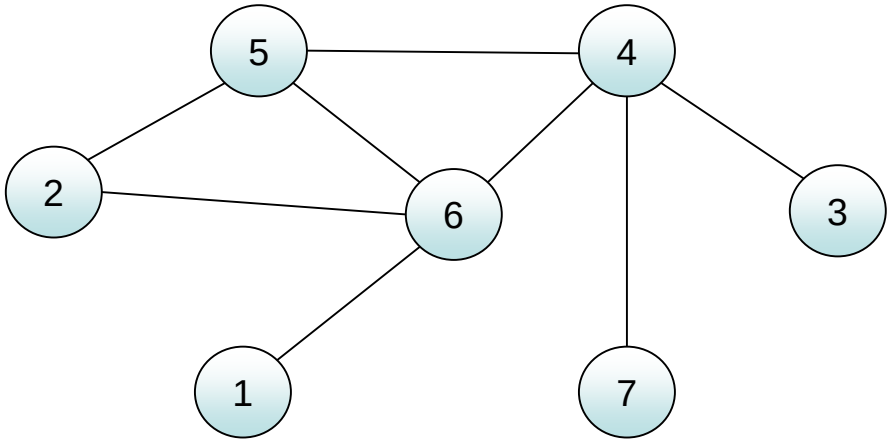
Bidirectional ring leader election (HS)

```
HSRingLeaderElection:   #  vstup myid > 0, in, out; výstup: status – leader alebo not leader

status = not leader; decide = false; range = 1; echo = 0
send(myid,in,range); send(myid,out,range)

while not decide :
    receive(i, dir, tag)      #  dir .. direction  in = 0    out = 1

    if  i > myid :
            if  tag > 1 :  send(i, 1-dir, tag-1)    # 1-dir je opačné rozhranie k dir
            if  tag == 1 : send(i, dir, 0)
            if  tag == 0 : send(i, 1-dir, 0)
    if i == myid :
            if echo == 0 : echo = 1          #  očakáva potvrdenie z oboch strán
            elif (echo == 1) and (tag > 0) :
                  status = leader; send(0, out, myid)    # terminácia
            elif (echo == 1) and (tag == 0) :        # prechod do novej fázy
                   echo = 0; range *= 2; send(myid, in, range); send(myid, out, range)
    if i == 0 :
            decide = true
            if status <> leader : send(0, out, tag)
```

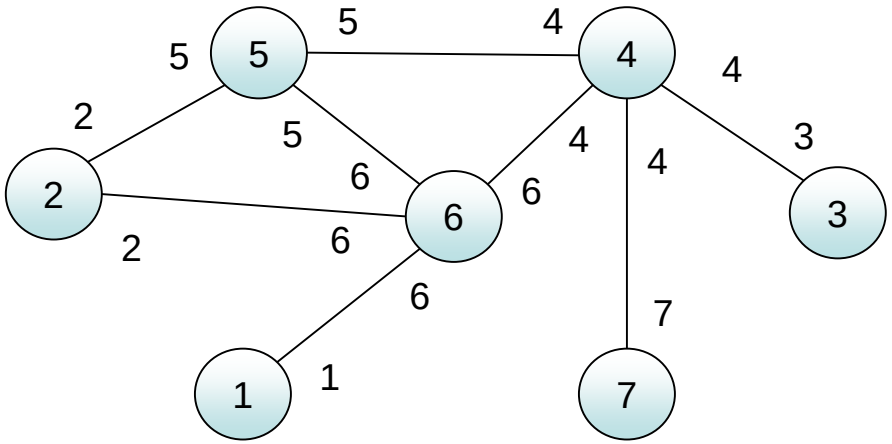
Voľba koordinátora záplavou

- záplava (flooding) – rozosielanie správ všetkými smermi
- proces na začiatku označí svoje myid za maximálne
- v každom kroku prepošle maximálne id a prijme správy, z ktorých určí nové maximum
- problém ukončenia ...
- ak poznám d - priemer grafu (maximum z najkratších dĺžok ciest medzi všetkými dvojicami vrcholov)
- ak poznám počet procesov n – komunikačná zložitosť $2.|E|.n \in O(n^3)$

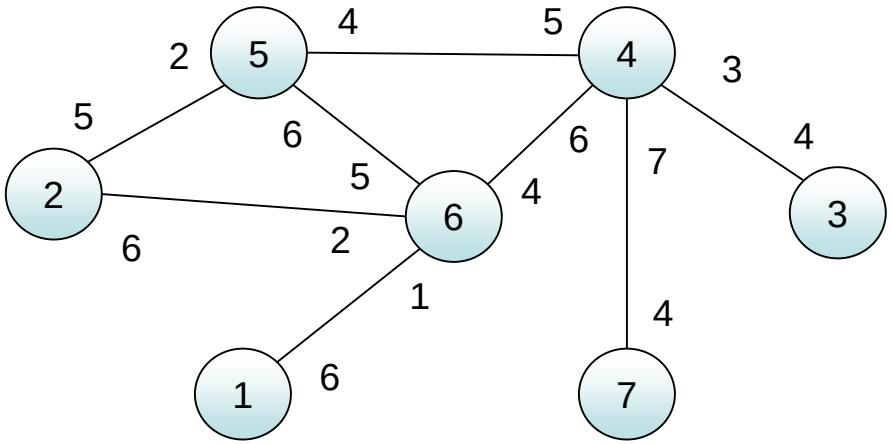
Voľba koordinátora záplavou



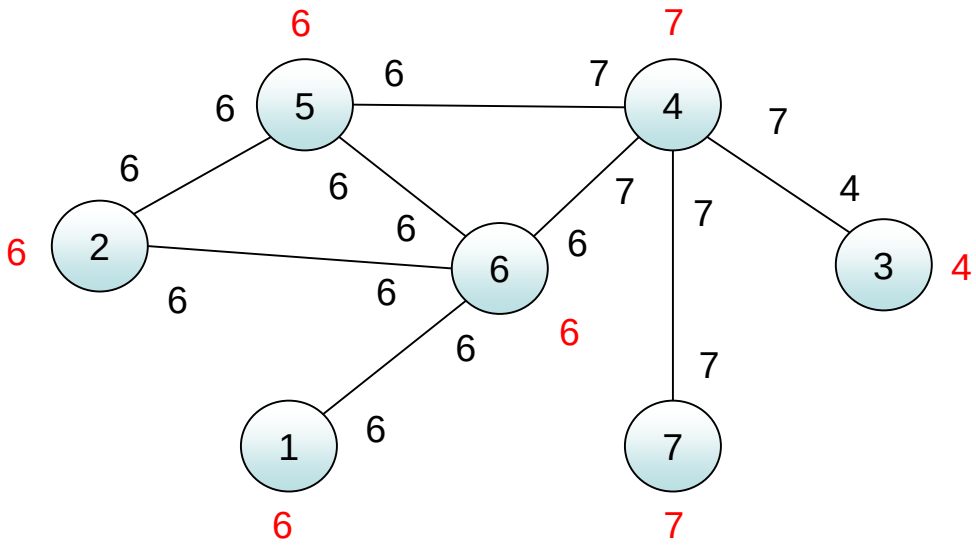
Voľba koordinátora záplavou



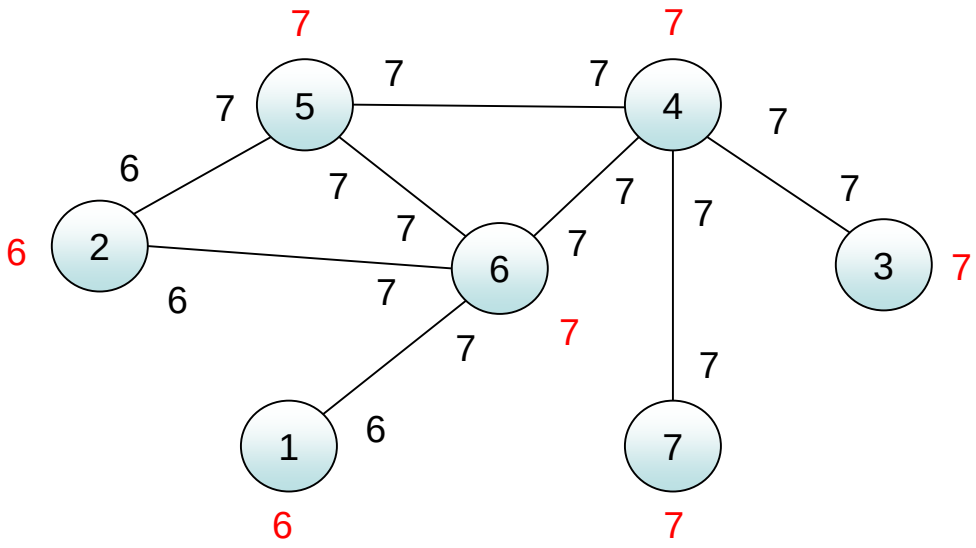
Voľba koordinátora záplavou



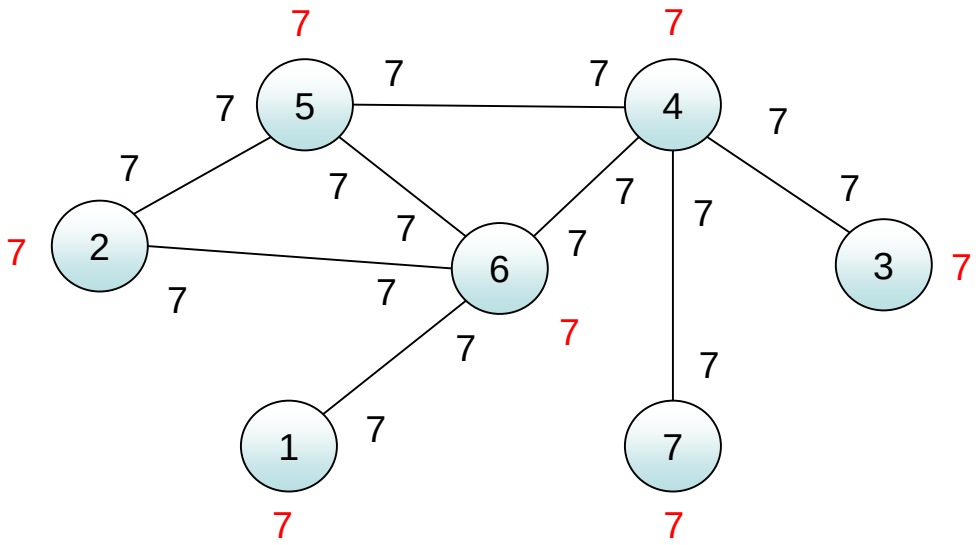
Voľba koordinátora záplavou



Voľba koordinátora záplavou



Voľba koordinátora záplavou



Voľba koordinátora záplavou

procedure FloodMax

{vstup: S – susedné procesy, diam – priemer siete; výstup: status – leader or not leader}

status := **not** leader; maxid := **myid;**

for rounds := 1 **to** diam **do**

forall v **in** S **do send**(maxid, v, 0) **endforall;**

forall v **in** S **do receive**(id, v, 0); **if** id > maxid **then** maxid := id **endif endforall**

endfor;

if maxid = **myid then** status := leader **endif**

end FloodMax

{veľká komunikačná zložitosť – diam*m ~ $O(n^3)$, nutnosť poznať maximálnu vzdialenosť procesov, alebo aspoň počet vrcholov}

Voľba koordinátora záplavou - zlepšenie

procedure FloodMax2 {ďalšia vlna záplavy sa posiela len ak došlo k zmene}

{vstup: S – susedné procesy, diam – priemer siete; výstup: status – leader}

new := **true**; status := **not** leader; maxid := **mysids;**

for rounds := 1 **to** diam **do**

if new **then** new := **false; forall** v **in** S **do send**(maxid, v, 0) **endforall**
else send(maxid, **myid**, 0) **endif;** { správu pošle len sebe }

while probe(ANY_SOURCE, 0) **do** { pre všetky správy, prijaté v jednom kroku}
receive(id, v, 0); **if** id > maxid **then** maxid := id; new := **true endif**
endwhile;

endfor;

if maxid = **myid then** status := leader **endif;**

end FloodMax2 { kvôli synchronizácii sa správa po každom kroku pošle aspoň sebe,
na príjem v jednom kroku nemusím dostať správy od všetkých susedov - test **probe** }

Záplavové algoritmy (flood algorithms)

Záplavové algoritmy

- úplne distribuované (neexistuje koordinátor)
- možno vyvolať v pravidelných intervaloch, resp. pri zmene topológie siete
- on-line informácie – možnosť eliminácie chýb (STP protokol, RIP protokol v TCP/IP)
- nevýhoda: veľký počet správ

Algoritmy vlny

- princíp vlny – od koordinátora (iniciátora)
- správa sa nepreposiela do smeru, odkiaľ prišla
- je výhodné vopred stanoviť priebeh vlny – orientovaná kostra s koreňom – iniciátorom
- možno očakávať odpoveď (echo) v opačnom smere ako smer pôvodne šírenej vlny (od listov ku koreňu)
- broadcast – convergecast (echo)

Šírenie správ v stromovej sieti vlnou po šírke

procedure TreeBFS { Breadth-first Search }

{inp: S – susedné procesy, init – iniciátor; out: p – rodičovský proces}

if myid = init **then** p := **myid else receive**(0, p, join) **endif;**

forall v **in** S - {p} **do send**(0, v, join) **endforall**

end TreeBFS

šírenie správ vlnou, pokiaľ je vytvorená usmernená (resp. obojsmerná) sieť

- vytvorenie logického kruhu
- vytvorenie logického stromu usmerneného ku koreňu (**koreň je sám sebe rodičom**)

zložitosť – čas úmerný diametru, komunikácia (n-1) správ

možno použiť na broadcast (napr. ako signál na ukončenie všetkých procesov)

Prehľadávanie stromovej siete so spätnou väzbou

procedure TreeBFSecho { BFS s echom broadcast-convergecast}

{inp: S – susedné procesy, init – iniciátor; out: p – rodičovský proces}

if myid = init **then** p := **myid else receive**(0, p, join) **endif;**

forall v **in** S - {p} **do send**(0, v, join) **endforall;**

forall v **in** S - {p} **do receive**(1, v, echo) **endforall;**

if myid <> p **then send**(1, p, echo) **endif**

end TreeBFSecho

požadované údaje je možné získať cestou od listov ku koreňu
(koreň má ako rodiča samého seba)

- výpočty zo stavov jednotlivých procesov
- voľba koordinátora, výpočet priemeru (diametra)

Prehľadávanie stromovej siete so spätnou väzbou

procedure TreeBFSecho { BFS s echom broadcast-convergecast}

{inp: S – susedné procesy, init – iniciátor; out: p – rodičovský proces}

if myid = init **then** p := **myid else receive**(0, p, join) **endif;**

forall v **in** S - {p} **do send**(0, v, join) **endforall;** U := ø;

while U <> S - {p} **do receive**(1, v, echo); U := U + {v} **endwhile;**

if myid <> p **then send**(1, p, echo) **endif**

end TreeBFSecho

požadované údaje je možné získať cestou od listov ku koreňu
(koreň má ako rodiča samého seba)

- výpočty zo stavov jednotlivých procesov
- voľba koordinátora, výpočet priemeru (diametra)

```
procedure TreeSearch
{inp: S – susedné procesy, init – iniciátor, search – kľúč
    k hľadanému objektu (volí iniciátor); out: p – rodičovský proces,
    sid – index procesu, vlastniaceho objekt alebo 0}
if myid = init then p := myid  else receive(search, p, join) endif;
sid := 0; if search in MyObj then sid := myid
   else forall v in S - {p} do send(search, v, join) endforall;
   forall v in S - {p} do receive(resp, v, echo);
                  if resp<>0 then sid := resp endif endforall endif;
if myid <> p then send(sid, p, echo) endif
end TreeSearch
```

Zastavenie a odrazenie vlny

Zastavenie vlny - ak sa často dosahuje výsledok skôr, ako vlna dorazí k listom

- prvotnú výzvu (join) preposielajú uzly spomalene (napr. medzi príjem a odoslanie vložia stále jeden prázdny krok napr. správu sebe)
- keď výzva dosiahne cieľ – zastaví cieľový proces prvotnú výzvu vlnou na zrušenie (abort) bez spomaľovacích krokov
- resp. po dosiahnutí cieľa a príjme odpovede (echo) vyšle vlnu na zrušenie bez spomaľovacích krokov iniciátor (aby rušiaca vlna bola len jedna)

Grafovými algoritmami možno riešiť veľa globálnych problémov v distribuovanom systéme

- topológia siete
- najkratšia cesta k cieľu
- kostra siete
- maximálne toky medzi procesmi
- farbenie, maximálna nezávislá množina ...

Konštrukcia BFS stromu vo všeobecnej sieti

procedure BFS {paralelné prehľadávanie siete do šírky}

{inp: S – susedné procesy, init – iniciátor; out: p – rodičovský proces v strome}

status := **not** tree; nrep := 0; { počet odpovedí ys (your son) a nys }

if myid = init **then**
p := **myid**; status := tree; **forall** u **in** S **do send**(0, u, join) **endforall endif**;

{pokiaľ počet prijatých správ ys a nys je menej, ako počet susedov}

while nrep < #S **do receive**(0, v, tag);

if tag = join **then if** status = tree **then send**(0, v, nys) {ďalší join odmietnuť}
else status := tree; p := v; nrep := nrep + 1; { prvý join }
forall u **in** S - {p} **do send**(0, u, join) **endforall endif**
else nrep := nrep + 1 { za ys a nys }
endif

endwhile; **if myid** <> init **then send**(0, p, ys) **endif**

end BFS { čas úmerný diametru, správ najviac $4.|E| - 2n \in O(n^2)$ }

Prehľadávanie do hĺbky (Depth-first search)

```
procedure DFS {postupné prehľadávanie stromovej siete do hĺbky}
{inp: S – susedné procesy, init – iniciátor; out: p – rodičovský proces}
p := undef; used := {};  { used – množina vybavených susedov }
if myid = init then
    p := myid; choose u in S do send(tok, u, 0) endchoose; used := used + {u};
endif;
while (#S > #used)  do receive(tok, v, 0);  if p = undef then p := v endif;
    if (#S - #used) > 1 then
        choose u in S – ({p} + used) do used := used + {u}; send(tok,u,0)
        endchoose
    else used := used + {p}; send(tok, p, 0)
    endif
endwhile
end DFS        { traversal algorithm – začína a končí v init, prejde všetkými uzlami}
```

Ďakujem za pozornosť !