v sieti: posielanie správ (sokety)

server

```
int main(int argc, char *argv[]) {
    int sockfd, newsockfd, portno;
    socklen_t clilen;
    char buffer[256];
    struct sockaddr_in serv_addr, cli_addr;
    int n;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    bzero((char *) &serv_addr, sizeof(serv_addr));
    portno = atoi(argv[1]);

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);

    bind(sockfd, (struct sockaddr *) &serv_addr,
        sizeof(serv_addr));
    listen(sockfd,5);
    clilen = sizeof(cli_addr);
    newsockfd = accept(sockfd,
        (struct sockaddr *) &cli_addr, &clilen);
    bzero(buffer,256);

    n = read(newsockfd,buffer,255);
    printf("Here is the message: %s\n",buffer);

    n = write(newsockfd,"I got your message",18);
    close(newsockfd);
    close(sockfd);
 }
```

klient

```
int main(int argc, char *argv[]) {
    int sockfd, portno, n;
    struct sockaddr_in serv_addr;
    struct hostent *server;
    char buffer[256];

    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    server = gethostbyname(argv[1]);

    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,
    (char *)&serv_addr.sin_addr.s_addr,
    server->h_length);
    serv_addr.sin_port = htons(portno);

    connect(sockfd,(struct sockaddr *) &serv_addr,
        sizeof(serv_addr));

    printf("Please enter the message: ");
    bzero(buffer,256);
    fgets(buffer,255,stdin);
    n = write(sockfd,buffer,strlen(buffer));
    bzero(buffer,256);
    n = read(sockfd,buffer,255);
    printf("%s\n",buffer);
    close(sockfd);
    return 0;
}
```

v sieti: posielanie správ (MPI)

```
int main(int argc, char *argv[]) {
    const int tag = 47;
    ...
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &ntasks);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);

    if (id == 0) {
      MPI_Get_processor_name(msg, &length);
      printf("Hello World from process %d running on %s\n", id, msg);
      for (i=1; i<ntasks; i++) {
        MPI_Recv(msg, 80, MPI_CHAR, MPI_ANY_SOURCE,
                      tag, MPI_COMM_WORLD, &status);
        source_id = status.MPI_SOURCE;
        printf("Hello World from process %d running on %s\n", source_id, msg);
      }
    }
    else {
      MPI_Get_processor_name(msg, &length);
      MPI_Send(msg, length, MPI_CHAR, 0, tag, MPI_COMM_WORLD);
    }
    MPI_Finalize();
    if (id==0) printf("Ready\n");
}
```

abstraktný sieťový model

- nezávislé zariadenia (procesy, procesory, uzly)
- posielanie správ
- lokálny pohľad (číslovanie portov)
- asynchrónne, spoľahlivé správy
- topológia siete
- wakeup/terminácia

zložitosť: v závislosti od počtu procesorov

- počet správ / bitov
- čas (beh normovaný na max. dĺžku 1)

abstraktný sieťový model

horný odhad pre problém

Existuje algoritmus, ktorý **pre všetky** topológie, vstupy, časovania, ...
... pracuje správne a vymení najviac $f(n)$ správ

dolný odhad pre problém

Pre každý algoritmus, **existuje kombinácia** topológie, vstupu, časovania, ...
... že buď nepracuje správne alebo vymení aspoň $f(n)$ správ

prehľadávane

- na začiatku je jeden iniciátor
- treba informovať všetkých

shout-and-echo

- iniciátor pošle **shout**
- ak príde **shout** do nového vrchola
  - označí hranu
  - pošle po neoznačených **shout**
  - čaká na všetky **echo**
  - pošle **echo**
- ak príde **shout** do navštíveného – okamžite **echo**

prehľadávane

- na začiatku je jeden iniciátor
- treba informovať všetkých

shout-and-echo

- iniciátor pošle **shout**
- ak príde **shout** do nového vrchola
  - označí hranu
  - pošle po neoznačených **shout**
  - čaká na všetky **echo**
  - pošle **echo**
- ak príde **shout** do navštíveného – okamžite **echo**

zložitosť

- $4m$ správ
- $2diam(G)$ čas

traverzovanie

- na začiatku je jeden iniciátor
- treba informovať všetkých
- v každom okamihu je len jeden *token*

$f(x)$-traverzovanie

Na objavenie x vrcholov treba $\max\{f(x), n\}$ správ.

prehľadávanie do hĺbky

var $used_p[q]$: boolean **init** false for each $q \in Neigh_p$;
(* Indicates whether p has already sent to q *)
$father_p$: process **init** $udef$;

For the initiator only, execute once:
begin $father_p := p$; choose $q \in Neigh_p$;
$used_p[q] := true$; send $\langle$ **tok** $\rangle$ to q
end

For each process, upon receipt of $\langle$ **tok** $\rangle$ from q_0:
begin **if** $father_p = udef$ **then** $father_p := q_0$;
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 begin if $father_p \neq q_0 \wedge \neg used_p[q_0]$
then $q := q_0$
else choose $q \in Neigh_p \setminus \{father_p\}$
with $\neg used_p[q])$;
$used_p[q] := true$; send $\langle$ **tok** $\rangle$ to q
end
else begin $used_p[father_p] := true$;
send $\langle$ **tok** $\rangle$ to $father_p$
end
end

- zložitosť $2m$ (čas aj správy)
- ako ušetriť čas?

Awerbuch: $4m$ správ, $4n - 2$ čas

```
var used_p[q]   : boolean    init false for each q ∈ Neigh_p ;
                             (* Indicates whether p has already sent to q *)
    father_p    : process    init udef ;

For the initiator only, execute once:
    begin father_p := p ; choose q ∈ Neigh_p ;
          forall r ∈ Neigh_p do send ⟨vis⟩ to r ;
          forall r ∈ Neigh_p do receive ⟨ack⟩ from r ;
          used_p[q] := true ; send ⟨tok⟩ to q
    end

For each process, upon receipt of ⟨tok⟩ from q_0:
    begin if father_p = udef then
            begin father_p := q_0 ;
                  forall r ∈ Neigh_p \ {father_p} do send ⟨vis⟩ to r ;
                  forall r ∈ Neigh_p \ {father_p} do receive ⟨ack⟩ from r
            end ;
          if p is the initiator and ∀q ∈ Neigh_p : used_p[q]
            then decide
          else if ∃q ∈ Neigh_p : (q ≠ father_p ∧ ¬used_p[q])
            then begin if father_p ≠ q_0 ∧ ¬used_p[q_0]
                         then q := q_0
                         else choose q ∈ Neigh_p \ {father_p}
                                    with ¬used_p[q] ;
                       used_p[q] := true ; send ⟨tok⟩ to q
                 end
            else begin used_p[father_p] := true ;
                       send ⟨tok⟩ to father_p
                 end
    end

For each process, upon receipt of ⟨vis⟩ from q_0:
    begin used_p[q_0] := true ; send ⟨ack⟩ to q_0 end
```

- **otázka:** treba čakať na **ack**?

prehľadávanie do šírky

Cheung'83: kubická komunikácia, lineárny čas

- každá správa obsahuje počítadlo hopov
- na začiatku iniciátor: všetkým susedom pošle 1
- každý vrchol p: lokálnu vzdialenosť $dist_p := \infty$
- on recv i: $dist_p := \min\{i, dist_p\}$, ak sa zmenila, pošli všetkým

Cheung,Zhu'87: kvadratická komunikácia aj čas

- buduje kostru po vrstvách

kombinácia?

voľba šéfa: úplné grafy

const: N : integer
ID : integer
$Neigh$: [1...N-1] link
var: $leader$: boolean
$count$: integer
i : integer

Init:

$count := 0$
$leader :=$ **false**

Code:

for $i = 1$ **to** $N - 1$ **do**
    send $\langle$**elect**, $ID\rangle$ **to** $Neigh[i]$
while $count < N - 1$ **wait**
for $i = 1$ **to** $N - 1$ **do**
    send $\langle$**leader**, $ID\rangle$ **to** $Neigh[i]$
$leader :=$ **true**

On receipt $\langle$**elect**, $id_i\rangle$ from $Neigh[i]$:

if $id_i > ID$ **send** $\langle$**accept**$\rangle$ **to** $Neigh[i]$

On receipt $\langle$**accept**$\rangle$ from $Neigh[i]$:

$count++$

On receipt $\langle$**leader**, $id_i\rangle$ from $Neigh[i]$:

Skonči algoritmus

voľba šéfa: úplné grafy II

const: N : integer
ID : integer
$Neigh$: [1...N-1] link
var: $leader$: boolean
$state$: {active, captured, killed}
$level$: integer
$parent$: link
msg : {victory, defeat}
i : integer

Init:

$state := \text{active}$
$level := 0$
$leader :=$ **false**

Code:

for $i = 1$ **to** $N - 1$ **do**
  send $\langle$**capture**, $[level, ID]\rangle$ **to** $Neigh[i]$
  receive $\langle$**accept**$\rangle$ **from** $Neigh[i]$
  $level++$
$leader :=$ **true**
for $i = 1$ **to** $N - 1$ **do**
  send $\langle$**leader**, $ID\rangle$ **to** $Neigh[i]$

Dead:

loop forever

On receipt $\langle$**capture**, $[level_i, id_i]\rangle$ from $Neigh[i]$:

if $state \in \{\text{active}, \text{killled}\}$ **and** $[level_i, id_i] > [level, ID]$
  $state := \text{captured}$
  $parent := Neigh[i]$
  send $\langle$**accept**$\rangle$ **to** $parent$
  goto Dead
else if $state = \text{captured}$
  send $\langle$**help**, $[level_i, id_i]\rangle$ **to** $parent$
  receive msg **from** $parent$
  if $msg = \text{defeat}$
    send $\langle$**accept**$\rangle$ **to** $Neigh[i]$
    $parent := Neigh[i]$

On receipt $\langle$**help**, $[level_i, id_i]\rangle$ from $Neigh[i]$:

if $[level_i, id_i] < [level, ID]$
  send $\langle$victory$\rangle$ **to** $Neigh[i]$
else
  send $\langle$defeat$\rangle$ **to** $Neigh[i]$
  if $state = \text{active}$
    $state := \text{killed}$
    goto Dead

On receipt $\langle$**leader**, $id_i\rangle$ from $Neigh[i]$:

Skonči algoritmus

voľba šéfa: úplné grafy II - analýza

Lema 1

V ľubovoľnom výpočte existuje pre každý level $l = 0, ..., N-1$ aspoň jeden proces, ktorý bol počas výpočtu na leveli l.

Lema 2

Nech v je aktívny proces ($state = \mathrm{active}$) s levelom l. Potom existuje l zajatých procesov ktoré patria v (t.j. ich premenná *parent* ukazuje na v).

$\Rightarrow$ práve jeden proces je šéf

Lemma 3

ľubovoľnom výpočte je najviac $N/(l+1)$ procesov, ktoré niekedy dosiahli level l.

$\Rightarrow$ maximálne $\sum_{l=1}^{N-1} \frac{N}{l+1} = N(\mathbf{H}_N - 1) \approx N \log N$ postupov o level (=správ)

dolný odhad

Treba $\Omega(n \log n)$ správ

- "globány" algoritmus
- graf indukovaný novými správami
- udržujem výpočet:
  - jeden súvislý komponent, ostatné vrcholy izolované
  - $e(k)$ – koľko viem vynútiť na komponente k
  - $e(2k+1) = 2e(k) + k + 1$

jednosmerné kruhy

Chang, Roberts

const: ID : integer
l_{in}, l_{out} : link
var: $leader$: integer

Init:

$leader := NULL$

Code:

send $\langle ID \rangle$
wait until $leader <> NULL$

On receipt $\langle i \rangle$:

if $i < ID$ **then send** $\langle i \rangle$
if $i = ID$ **then**
  $leader := ID$
  send $\langle \textbf{leader}, ID \rangle$

On receipt $\langle \textbf{leader}, x \rangle$:

$leader := x$
send $\langle \textbf{leader}, ID \rangle$

- $\Omega(n^2)$ správ v najhoršom prípade
- $O(n \log n)$ v priemernom

Chang Roberts - analýza

Koľkokrát sa pohla správa i?

$$P(i,k) = \frac{\binom{n-i}{k-1} \cdot (i-1)}{\binom{n-1}{k-1} \cdot (n-k)}$$

$$E[X_i] = \sum_{k=1}^{n-i+1} k \cdot P(i,k)$$

$$E[X] = n + \sum_{i=2}^{n} E[X_i] = n + \sum_{i=2}^{n} \sum_{k=1}^{n-i+1} k \cdot P(i,k)$$

Chang Roberts - analýza

lema

$$\sum_{j=k}^{n-1} \frac{j!}{(j-k)!} = k! \binom{n}{k+1}$$

dôkaz

$$\sum_{j=k}^{n-1} \frac{j!}{(j-k)!} = \sum_{j=k}^{n-1} \frac{k!j!}{k!(j-k)!} = k! \sum_{j=k}^{n-1} \binom{j}{k} = k! \binom{n}{k+1}$$

$$n+\sum_{i=2}^{n}\sum_{k=1}^{n-i+1} k\cdot\frac{\binom{n-i}{k-1}\cdot(i-1)}{\binom{n-1}{k-1}\cdot(n-k)} = n+\sum_{j=1}^{n-1}\sum_{k=1}^{j} k\cdot\frac{\binom{j-1}{k-1}\cdot(n-j)}{\binom{n-1}{k-1}\cdot(n-k)} =$$

$$= n+\sum_{j=1}^{n-1}\sum_{k=1}^{j}\frac{k(j-1)!(n-j)(k-1)!(n-k)!}{(k-1)!(j-k)!(n-1)!(n-k)} =$$

$$= n+\sum_{k=1}^{n-1}\left[\frac{k(n-k-1)!}{(n-1)!}\sum_{j=k}^{n-1}\frac{(j-1)!(n-j)}{(j-k)!}\right] =$$

$$= n+\sum_{k=1}^{n-1}\left[\frac{k(n-k-1)!}{(n-1)!}\left(\sum_{j=k}^{n-1}\frac{n(j-1)!}{(j-k)!}-\sum_{j=k}^{n-1}\frac{j!}{(j-k)!}\right)\right] =$$

$$\sum_{j=k}^{n-1}\frac{(j-1)!}{(j-k)!} = (k-1)!\binom{n-1}{k} \text{ a } \sum_{j=k}^{n-1}\frac{j!}{(j-k)!} = k!\binom{n}{k+1}$$

$$= n+\sum_{k=1}^{n-1}\frac{k(n-k-1)!}{(n-1)!}\left[n\cdot(k-1)!\binom{n-1}{k}-k!\binom{n}{k+1}\right] =$$

$$= n+\sum_{k=1}^{n-1}\frac{n}{k+1}$$

dvojsmerné kruhy

Hirschberg-Sinclair

- level l: dobýjať územie 2^l
- $\log n$ levelov
- $n/2^l$ vrcholov na leveli
- každý vrchol pošle 2^l správ

dvojsmerné kruhy

Hirschberg-Sinclair

- level l: dobýjať územie 2^l
- $\log n$ levelov
- $n/2^l$ vrcholov na leveli
- každý vrchol pošle 2^l správ

Franklin

- level l: poraziť susedov (na rovnakom leveli; "synchronizácia")
- $\log n$ levelov
- n správ na level

dvojsmerné kruhy

Hirschberg-Sinclair

- level l: dobýjať územie 2^l
- $\log n$ levelov
- $n/2^l$ vrcholov na leveli
- každý vrchol pošle 2^l správ

Franklin

- level l: poraziť susedov (na rovnakom leveli; "synchronizácia")
- $\log n$ levelov
- n správ na level

Dolev – simulácia na 1-smernom kruhu

- idea: presunúť identitu

dolný odhad

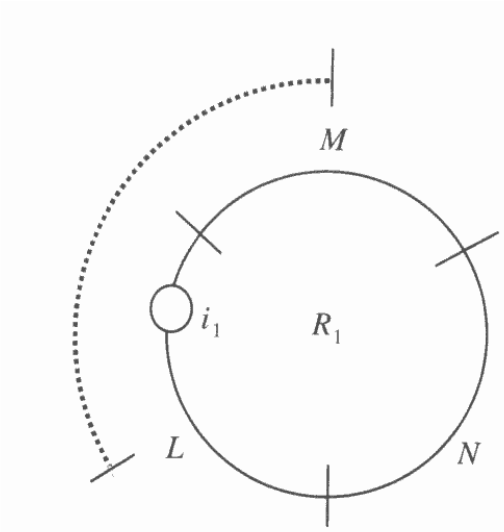
zapojiť do čiary L, vymení sa $C(L)$ správ

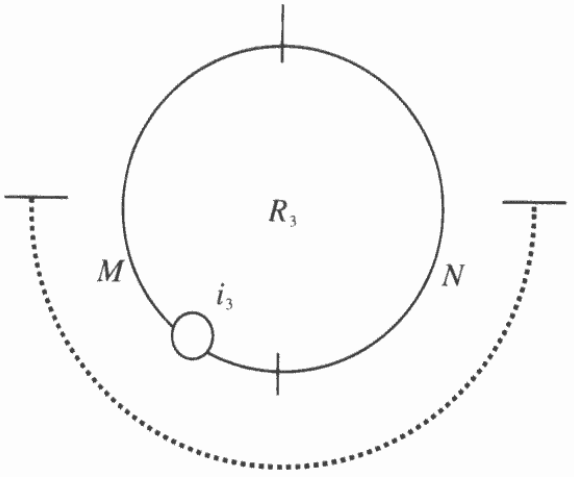
lema

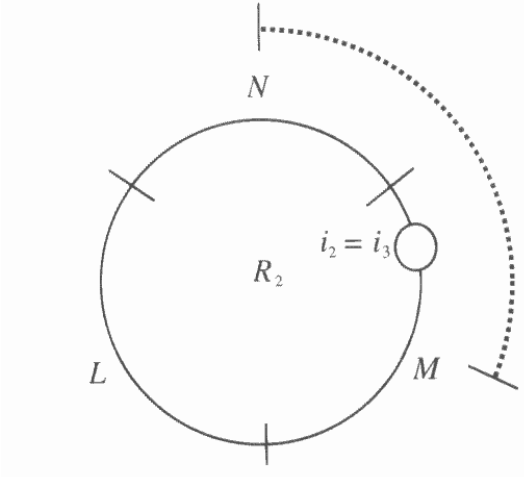
Pre každé r existuje nekonečne veľa čiar dĺžky 2^r, kde $C(L) \geq r2^{r-2}$

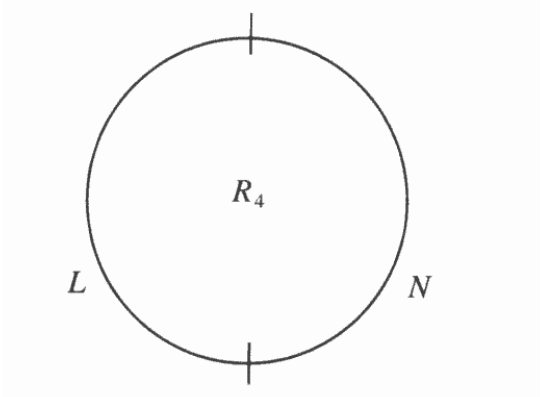
- indukcia
- dve vrecia: vyberám trojice, chcem dve spojiť item pri spojení: dĺžka 2^{r+1}, počet správ $\geq r2^{r-1}$
- ešte treba 2^{r-1} správ; sporom

dolný odhad









GHS

- ľubovoľná topológia
- buduje sa kostra
- "segmenty"
- spájanie po najlacnejšej odchádzajúcej hrane:
  - A menší sa pripojí k väčšiemu
  - B rovnakí sa spoja
  - C väčší čaká
- veľkosť $\Rightarrow$ level

GHS

var $state_p$: $(sleep, find, found)$;
$stach_p[q]$: $(basic, branch, reject)$ for each $q \in Neigh_p$;
$name_p$, $bestwt_p$: real ;
$level_p$: integer ;
$testch_p$, $bestch_p$, $father_p$: $Neigh_p$;
rec_p : integer ;

(1) As the first action of each process, the algorithm must be initialized:
begin let pq be the channel of p with smallest weight ;
$stach_p[q] := branch$; $level_p := 0$;
$state_p := found$; $rec_p := 0$;
send $\langle$ **connect**, 0 $\rangle$ to q
end

(2) Upon receipt of $\langle$ **connect**, L $\rangle$ from q:
begin if $L < level_p$ **then** (* Combine with Rule A *)
begin $stach_p[q] := branch$;
send $\langle$ **initiate**, $level_p$, $name_p$, $state_p$ $\rangle$ to q
end
else if $stach_p[q] = basic$
then (* Rule C *) process the message later
else (* Rule B *) send $\langle$ **initiate**, $level_p + 1, \omega(pq), find$ $\rangle$ to q
end

(3) Upon receipt of $\langle$ **initiate**, L, F, S $\rangle$ from q:
begin $level_p := L$; $name_p := F$; $state_p := S$; $father_p := q$;
$bestch_p := udef$; $bestwt_p := \infty$;
forall $r \in Neigh_p : stach_p[r] = branch \wedge r \neq q$ **do**
send $\langle$ **initiate**, L, F, S $\rangle$ to r ;
if $state_p = find$ **then begin** $rec_p := 0$; $test$ **end**
end

GHS

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

(5) Upon receipt of $\langle$ **test**, L, F $\rangle$ from q:
begin if $L > level_p$ **then** (* Answer must wait! *)
process the message later
else if $F = name_p$ **then** (* internal edge *)
begin if $stach_p[q] = basic$ **then** $stach_p[q] := reject$;
if $q \neq testch_p$
then send $\langle$ **reject** $\rangle$ to q
else *test*
end
else send $\langle$ **accept** $\rangle$ to q
end

(6) Upon receipt of $\langle$ **accept** $\rangle$ from q:
begin $testch_p := udef$;
if $\omega(pq) < bestwt_p$
then begin $bestwt_p := \omega(pq)$; $bestch_p := q$ **end** ;
report
end

(7) Upon receipt of $\langle$ **reject** $\rangle$ from q:
begin if $stach_p[q] = basic$ **then** $stach_p[q] := reject$;
test
end

GHS

(8) **procedure** *report*:
begin if $rec_p = \#\{q : stach_p[q] = branch \wedge q \neq father_p\}$
and $testch_p = udef$ **then**
begin $state_p := found$; send $\langle$ **report**, $bestwt_p$ $\rangle$ to $father_p$ **end**
end

(9) Upon receipt of $\langle$ **report**, ω $\rangle$ from q:
begin if $q \neq father_p$
then (* reply for **initiate** message *)
begin if $\omega < bestwt_p$ **then**
begin $bestwt_p := \omega$; $bestch_p := q$ **end** ;
$rec_p := rec_p + 1$; *report*
end
else (* pq is the 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**
end

(10) **procedure** *changeroot*:
begin if $stach_p[bestch_p] = branch$
then send $\langle$ **changeroot** $\rangle$ to $bestch_p$
else begin send $\langle$ **connect**, $level_p$ $\rangle$ to $bestch_p$;
$stach_p[bestch_p] := branch$
end
end

(11) Upon receipt of $\langle$ **changeroot** $\rangle$:
begin *changeroot* **end**

analýza

správnosť

ukázať, že sa zvolí práve jeden šéf: nenastane deadlock

počet správ

- testovacie správy: jeden test po každej hrane
- kostrové správy: fragment s n_i vrcholmi pri postupe o level $O(n_i)$ správ
- postupy na level l: dizjunktné vrcholy

KKM

- $f(x)$-traverzovanie
- tokeny traverzujú/označujú územia
- levely: keď sa stretnú dva, vznikne nový
- naháňanie

KKM

```
var lev_p          : integer     init −1 ;
    cat_p, wait_p  : P           init udef ;
    last_p         : Neigh_p     init udef ;

begin if p is initiator then
        begin lev_p := lev_p + 1 ; last_p := trav(p, lev_p) ;
              cat_p := p ; send ⟨ annex, p, lev_p ⟩ to last_p
        end ;
      while ... (* Termination condition, see text *) do
        begin receive token (q, l) ;
              if token is annexing then t := A else t := C ;
              if l > lev_p then (* Case I *)
                begin lev_p := l ; cat_p := q ;
                      wait_p := udef ; last_p := trav(q, l) ;
                      send ⟨ annex, q, l ⟩ to last_p
                end
              else if l = lev_p and wait_p ≠ udef then (* Case II *)
                begin wait_p := udef ; lev_p := lev_p + 1 ;
                      last_p := trav(p, lev_p) ; cat_p := p ;
                      send ⟨ annex, p, lev_p ⟩ to last_p
                end
              else if l = lev_p and last_p = udef then (* Case III *)
                wait_p := q
              else if l = lev_p and t = A and q = cat_p then (* Case IV *)
                begin last_p := trav(q, l) ;
                      if last_p = decide
                        then p announces itself leader
                        else send ⟨ annex, q, l ⟩ to last_p
                end
              else if l = lev_p and ((t = A and
                   q > cat_p) or t = C) then (* Case V *)
                begin send ⟨ chase, q, l ⟩ to last_p ; last_p := udef end
              else if l = lev_p then (* Case VI *)
                wait_p := q
```

KKM

počet správ

- naháňacie: 1 na vrchol a level, spolu n za level
- objavovacie: $\sum_i f(n_i)$
- ak f je konvexná, t.j. $f(a) + f(b) \leq f(a + b)$, tak je $O(\log n(n + f(n)))$ správ