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

K_n: vplyv orientácie

zmysel pre orientáciu

Porty sú číslované podľa vzdialenosti vo fixnej Hamiltonovej kružnici.

- algoritmus: zajať fixný pattern $\{i[1..k], i[2k], i[3k], \ldots, i[n-k]\}$
- prvá fáza $i[1..k]$
  - *level*, *ID*
  - *level* je počet zajatých
  - pripája sa celé územie
- druhá fáza $i[2k], i[3k], \ldots, i[n-k]$
  - nastav *owner* vrcholom $i[1..k]$, ack
  - pošli *elect* do $i[2k], i[3k], \ldots, i[n-k]$
  - *elect*: ak je v prvej fáze, alebo je slabší $\Rightarrow$ accept

K_n: vplyv orientácie

počet správ

- prvá fáza: $O(n)$: odpoveď zajatému 1x, dizjunknosť území
- druhá fáza: max $O(n/k)$ kandidátov, každý pošle n/k správ $\Rightarrow$ $O(n^2/k^2)$ správ

čas

- po zobudení (najsilnejšieho) $O(k)$, v najhoršom $O(n)$
- pridať wakeup fázu (poslať od $i[1], i[k]$) $\Rightarrow$ $O(k + n/k)$
- $k := \sqrt{n}$

vplyv synchrónnosti

algoritmy založené na porovnaniach

- ekvivalentné okolia
- c-symetrický reťazec: pre každé $\sqrt{n} \leq l \leq n$ a pre každý segment S je $\left\lfloor \frac{cn}{l} \right\rfloor$ ekvivalentných
- bit-reversal je $1/2$-symetrický
- c-symetrický reťazec, alg. nemôže skončiť po k kolách pre $\left\lfloor \frac{cn}{2k+1} \right\rfloor \geq 2$
- počet správ: $k = \left\lfloor \frac{cn-2}{4} \right\rfloor$, je aspoň $k+1$ kôl
- aspoň $\left\lfloor \frac{cn}{2r-1} \right\rfloor$ aktívnych v r-tom kole pošle správu

vplyv synchrónnosti

algoritmy využívajúce integer ID

- rôzne rýchlosti
- v i-tom kroku test

cvičenie

V toruse rozmerov $n \times n$ (t.j. zacyklenej mriežke) sú na začiatku zobudené dva vrcholy. Napíšte algoritmus, pomocou ktorého sa každý z nich dozvie identifikátor druhého s použitím $O(n)$ správ.
Ako sa úloha zmení, ak je komunikačná sieť hyperkocka?

Nájdite asymptoticky optimálny (čo do počtu správ) algoritmus na voľbu šéfa v úplnom bipartitnom grafe $K_{n,n}$. Dokážte jeho zložitosť a optimalitu.

Nájdite algoritmus na voľbu šéfa v k-chordálnom kruhu s použitím $O(n + \frac{n}{k} \log \frac{n}{k})$ správ.

$\star$

broadcasting a voľba šéfa na (ne?)orientovanej hyperkocke s lineárnym počtom správ

routovanie

cieľ: doručovať srávy medzi ľubovoľnou dvojicou

modely

- destination based
- splittable
- connections (wormhole)
- buffering
- selfish

ciele

- statické váhy (najkratšie cesty)
- dynamické váhy (hot potato)
- deadlock

najkratšie cesty

```
begin (* Initialize S to ∅ and D to ∅-distance *)
        S := ∅ ;
        forall u, v do
                if u = v then D[u, v] := 0
                else if uv ∈ E then D[u, v] := ω_uv
                else D[u, v] := ∞ ;
        (* Expand S by pivoting *)
        while S ≠ V do
           (* Loop invariant: ∀u, v : D[u, v] = d^S(u, v) *)
           begin pick w from V \ S ;
                   (* Execute a global w-pivot *)
                   forall u ∈ V do
                           (* Execute a local w-pivot at u *)
                           forall v ∈ V do
                                   D[u, v] := min ( D[u, v], D[u, w] + D[w, v] ) ;
                   S := S ∪ {w}
           end     (* ∀u, v : D[u, v] = d(u, v) *)
end
```

najkratšie cesty

```
var S_u   : set of nodes ;
    D_u   : array of weights ;
    Nb_u  : array of nodes ;

begin S_u := ∅ ;
      forall v ∈ V do
            if v = u
              then begin D_u[v] := 0 ; Nb_u[v] := udef end
            else if v ∈ Neigh_u
              then begin D_u[v] := ω_uv ; Nb_u[v] := v end
            else begin D_u[v] := ∞ ; Nb_u[v] := udef end ;
      while S_u ≠ V do
        begin pick w from V \ S_u ;
              (* All nodes must pick the same node w here *)
              if u = w
                then "broadcast the table D_w"
                else "receive the table D_w"
              forall v ∈ V do
                    if D_u[w] + D_w[v] < D_u[v] then
                      begin D_u[v] := D_u[w] + D_w[v] ;
                            Nb_u[v] := Nb_u[w]
                      end ;
              S_u := S_u ∪ {w}
        end
end
```

najkratšie cesty

```
var S_u   : set of nodes ;
    D_u   : array of weights ;
    Nb_u  : array of nodes ;

begin S_u := ∅ ;
      forall v ∈ V do
            if v = u
              then begin D_u[v] := 0 ; Nb_u[v] := udef end
            else if v ∈ Neigh_u
              then begin D_u[v] := ω_uv ; Nb_u[v] := v end
            else begin D_u[v] := ∞ ; Nb_u[v] := udef end ;
      while S_u ≠ V do
         begin pick w from V \ S_u ;
               (* Construct the tree T_w *)
               forall x ∈ Neigh_u do
                     if Nb_u[w] = x then send ⟨ ys, w ⟩ to x
                                    else send ⟨ nys, w ⟩ to x ;
               num_rec_u := 0 ; (* u must receive |Neigh_u| messages *)
               while num_rec_u < |Neigh_u| do
                     begin receive ⟨ ys, w ⟩ or ⟨ nys, w ⟩ message ;
                           num_rec_u := num_rec_u + 1
                     end ;
               if D_u[w] < ∞ then (* participate in pivot round *)
                 begin if u ≠ w
                         then receive ⟨ dtab, w, D ⟩ fromthis Nb_u[w] ;
                       forall x ∈ Neigh_u do
                             if ⟨ ys, w ⟩ was received from x
                               then send ⟨ dtab, w, D ⟩ to x ;
                       forall v ∈ V do (* local w-pivot *)
                             if D_u[w] + D[v] < D_u[v] then
                               begin D_u[v] := D_u[w] + D[v] ;
                                     Nb_u[v] := Nb_u[w]
                               end
                 end ;
               S_u := S_u ∪ {w}
         end
end
```

Netchange

```
var Neigh_u      : set of nodes ;     (* The neighbors of u *)
    D_u          : array of 0.. N ;   (* D_u[v] estimates d(u, v) *)
    Nb_u         : array of nodes ;   (* Nb_u[v] is preferred neighbor for v *)
    ndis_u       : array of 0.. N ;   (* ndis_u[w, v] estimates d(w, v) *)

Initialization:
    begin forall w ∈ Neigh_u, v ∈ V do ndis_u[w, v] := N ;
          forall v ∈ V do
                begin D_u[v] := N ; Nb_u[v] := udef end ;
          D_u[u] := 0 ; Nb_u[u] := local ;
          forall w ∈ Neigh_u do send ⟨ mydist, u, 0 ⟩ to w
    end

Procedure Recompute (v):
    begin if v = u
            then begin D_u[v] := 0 ; Nb_u[v] := local end
            else begin (* Estimate distance to v *)
                       d := 1 + min{ndis_u[w, v] : w ∈ Neigh_u} ;
                       if d < N then
                         begin D_u[v] := d ;
                               Nb_u[v] := w with 1 + ndis_u[w, v] = d
                         end
                       else begin D_u[v] := N ; Nb_u[v] := udef end
                 end ;
          if D_u[v] has changed then
            forall x ∈ Neigh_u do send ⟨ mydist, v, D_u[v] ⟩ to x
    end
```

```
Processing a ⟨ mydist, v, d ⟩ message from neighbor w:
    { A ⟨ mydist, v, d ⟩ is at the head of Q_wv }
    begin receive ⟨ mydist, v, d ⟩ from w ;
          ndis_u[w, v] := d ; Recompute (v)
    end

Upon failure of channel uw:
    begin receive ⟨ fail, w ⟩ ; Neigh_u := Neigh_u \ {w} ;
          forall v ∈ V do Recompute (v)
    end

Upon repair of channel uw:
    begin receive ⟨ repair, w ⟩ ; Neigh_u := Neigh_u ∪ {w} ;
          forall v ∈ V do
                begin ndis_u[w, v] := N ;
                      send ⟨ mydist, v, D_u[v] ⟩ to w
                end
    end
```

Netchange

```
var Neigh_u   : set of nodes ;   (* The neighbors of u *)
    D_u       : array of 0.. N ; (* D_u[v] estimates d(u, v) *)
    Nb_u      : array of nodes ; (* Nb_u[v] is preferred neighbor for v *)
    ndis_u    : array of 0.. N ; (* ndis_u[w, v] estimates d(w, v) *)

Initialization:
  begin forall w ∈ Neigh_u, v ∈ V do ndis_u[w, v] := N ;
        forall v ∈ V do
              begin D_u[v] := N ; Nb_u[v] := udef end ;
        D_u[u] := 0 ; Nb_u[u] := local ;
        forall w ∈ Neigh_u do send ⟨ mydist, u, 0 ⟩ to w
  end

Procedure Recompute (v):
  begin if v = u
          then begin D_u[v] := 0 ; Nb_u[v] := local end
          else begin (* Estimate distance to v *)
                     d := 1 + min{ndis_u[w, v] : w ∈ Neigh_u} ;
                     if d < N then
                       begin D_u[v] := d ;
                             Nb_u[v] := w with 1 + ndis_u[w, v] = d
                       end
                     else begin D_u[v] := N ; Nb_u[v] := udef end
               end ;
        if D_u[v] has changed then
          forall x ∈ Neigh_u do send ⟨ mydist, v, D_u[v] ⟩ to x
  end
```

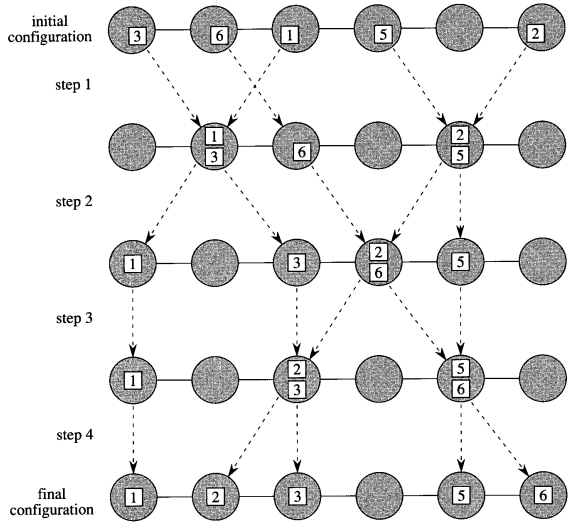
```
Processing a ⟨ mydist, v, d ⟩ message from neighbor w:
    { A ⟨ mydist, v, d ⟩ is at the head of Q_wv }
    begin receive ⟨ mydist, v, d ⟩ from w ;
          ndis_u[w, v] := d ; Recompute (v)
    end

Upon failure of channel uw:
    begin receive ⟨ fail, w ⟩ ; Neigh_u := Neigh_u \ {w} ;
          forall v ∈ V do Recompute (v)
    end

Upon repair of channel uw:
    begin receive ⟨ repair, w ⟩ ; Neigh_u := Neigh_u ∪ {w} ;
          forall v ∈ V do
                begin ndis_u[w, v] := N ;
                      send ⟨ mydist, v, D_u[v] ⟩ to w
                end
    end
```

korektnosť

lexikograficky klesá hodnota $[t_0, t_1, \ldots, t_N]$
kde t_i je počet správ $\langle \textbf{mydist}, i \rangle +$ počet dvojíc u, v kde $D_u[v] = i$

packet routing

- synchrónny režim
- vrcholy majú pakety (uložené v bufferoch)
- v jednom kroku po jednej linke ide max. jeden paket
- algoritmus = odchádzajúce linky + priorita bufferov
- celkový čas



packet routing na mriežke $\sqrt{N} \times \sqrt{N}$

Každý vrchol má 1 paket, do každého smeruje 1 paket (permutation routing)

Najprv riadok, potom stĺpec. Prednosť má ten s najdlhšou cestou.

analýza: stačí $2\sqrt{N} - 2$ krokov

- po $\sqrt{N} - 1$ krokoch je každý v správnom stĺpci (nebrzdia sa)
- routovanie v stĺpci ide v $\sqrt{N} - 1$ krokoch
  - pre každé i platí: po $N - 1$ krokoch sú koncové pakety na koncových miestach
  - dôvod: zdržujú sa iba navzájom

veľkosť buffra v najhoršom prípade: $2/3\sqrt{N} - 3$

veľkosť buffra: priemerný prípad I

setting

Každý vrchol má jeden paket s **náhodným cieľom**

max. veľkosť buffra $\approx$ počet zahnutí vo vrchole

psť, že aspoň r zahne $\leq \binom{\sqrt{N}}{r} \left(\frac{1}{\sqrt{N}}\right)^r < \left(\frac{e}{r}\right)^r$

pre $r = \frac{e \log N}{\log \log N}$ je psť $o(N^{-2})$

veľkosť buffra: priemerný prípad II

wide-channel: nepredbiehajú sa

lema

psť, že vo wch prejde aspoň $\alpha\Delta/2$ paketov cez hranu e počas $t+1, t+2, \ldots, t+\Delta$ je najviac $e^{(\alpha-1-\alpha \ln \alpha)\Delta/2}$

očakávaný počet paketov na hrane $(i,j) \mapsto (i+1,j)$ je

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

chceme ukázať, že s veľkou psťou ich neprejde príliš viac

Černovov odhad

lema

Majme n nezávislých Bernoulihho náh. prem. $X_1, \ldots, X_n$, pričom $Pr[X_k = 1] \leq P_k$. Potom

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

kde $X = \sum X_i$, $P = \sum P_i$

$$E[e^{\lambda X_k}] \leq 1 + P_k(e^{\lambda} - 1) \leq e^{P_k(e^{\lambda}-1)}$$

$$E[e^{\lambda X}] \leq e^{P(e^{\lambda}-1)}$$

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

veľkosť buffra: priemerný prípad II

lema

Majme n nezávislých Bernoulihho náh. prem. $X_1, \ldots, X_n$, pričom $Pr[X_k = 1] \leq P_k$. Potom

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

kde $X = \sum X_i$, $P = \sum P_i$

lema

psť, že vo wch prejde aspoň $\alpha\Delta/2$ paketov cez hranu e počas $t+1, t+2, \ldots, t+\Delta$ je najviac $e^{(\alpha-1-\alpha \ln \alpha)\Delta/2}$

očakávaný počet paketov na hrane $(i,j) \mapsto (i+1,j)$ je

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

chceme ukázať, že s veľkou psťou ich neprejde príliš viac

veľkosť buffra: priemerný prípad II

lema

ak je paket vo vzd. d od hrany e v čase T, a p prejde cez e v čase $T + d + \delta$, potom v každom kroku $[T + d, T + d + \delta]$ prejde paket cez e

dosledok

ak paket prejde cez e v čase T vo wch, a prejde cez e v čase $T + \delta$ v št. , tak v každom kroku $[T, T + \delta]$ prejde paket

lema

ak počas $[T + 1, T + \Delta]$ prejde cez e x paketov v št., tak pre nejaké t prejde $x + t$ paketov cez e v čase $[T + 1 = t, T + \Delta]$ vo wch.

lema

psť, že cez e prejde viac ako $\alpha\Delta/2$ paketov počas konkrétneho okna Δ krokov je najviac $O(e^{(\alpha-1-\alpha \ln \alpha)\Delta/2})$