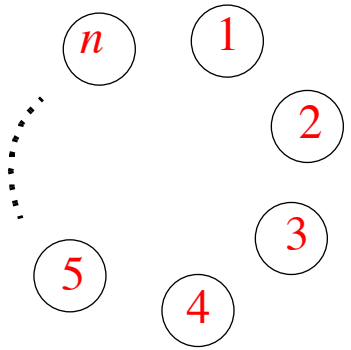
Eleição de Líder

O problema:

- Considere sistema G como anel de n vértices, numerados de 1 a n (mod n), no sentido horário.



- Premissa:
  - processos não conhecem seu índice ou de seus vizinhos, mas conseguem distinguir vizinho esquerdo do vizinho direito.
- Requisito:
  - Eventualmente (em algum ponto no futuro) apenas um processo se declara líder.

Eleição de Líder

Algumas variantes do problema:

- não líderes produzem saída indicando que não são líderes.
- anel uni- ou bi-direcional.
- valor de n pode ser conhecido ou não.
- processos podem ser idênticos ou podem ser distinguidos por identificador único.

Resultado de impossibilidade

Teorema: seja A um sistema de n processos, $n > 1$, em anel bidirecional. Se todos os processos em A são idênticos, então A não soluciona o problema de eleição.

Prova: por contradição. Suponha que exista A que soluciona o problema. Podemos assumir, sem perda de generalidade, que cada processo A tem exatamente um estado inicial (se este não é o caso, podemos produzir uma nova solução a partir de um único estado inicial para cada processo). Dessa forma, A tem apenas uma execução. Considere essa execução única. A prova prossegue por indução no número de turnos r já executados: totos os processos estão em estados idênticos após r turnos. Se um atinge o estado de líder, todos atingem ao mesmo tempo, o que é uma contradição.

Um algoritmo básico: LeLan (77), Chang e Roberts (79)

Variáveis:

- M = conjunto de uid's (identificadores únicos)
- Para cada i, $states_i$ consiste de
  - u = um uid, inicialmente o proprio uid de i
  - *send* = um uid ou nulo, inicialmente o próprio uid de i
  - *status* = {*desconhecido*, *líder*}, inicialmente *desconhecido*
- Conjunto de estados iniciais: um único estado com as inicializações dadas
- Função de geração de mensagens para cada i:

  envia valor corrente de *send* **para** processo $i + 1$ (mod n)
- Função de transição para cada i:

```
send := null;
if mensagem_recebida é v then
    case
        v > u: send := v
        v = u: status := líder
        v < u: nada a fazer
    endcase
```

Correção

Para provar correção, devemos mostrar que exatamente um processo produz saída *leader*. Seja i_{max} o índice do processo que tem o maior uid, e seja uid_{max} este uid. Precisamos provar que processo i_{max} produz saída *líder* e que nenhum outro processo produz essa saída.

Lema A: Processo i_{max} produz saída *líder* ao final do turno n.

Prova: Inicialmente note que as variáveis u nunca são alteradas (pelo código), têm inicialmente valores distintos, e i_{max} tem o maior valor de u (por definição de i_{max}), igual a u_{max}. Desta forma, basta provar que

A.1. Após n turnos, $status_{imax}$ = líder.

Para provar A.1, usamos uma asserção intermediária:

A.2. Para 0 < r < n-1, após r turnos, $send_{imax+r} = u_{max}$.

Esta asserção diz que o valor máximo, u_{max}, aparece na variável *send* do processo cuja posição no anel tem distância r de i_{max}. É fácil provar por indução: base é r=0, que é verdadeira pela inicialização. A prova do passo é baseada no fato de que todo nó diferente de i_{max} aceita o valor máximo e o atribui à variável *send*. Para completar a prova, basta examinar o que ocorre no último

turno: processo i_{max} recebe u_{max} na mensagem e com isto altera seu estado para *líder*.

Economizando Mensagens - Hirschberg e Sinclair (80)

Decentralized extrema-fiding in circular configurations of processors, D. S. Hirschberg, J. B. Sinclair. Communications of the ACM, novembro, 1980.

Variáveis:

- M = conjunto de uid's (identificadores únicos)
- Para cada i, $states_i$ consiste de
  - u = conjunto de triplas (UID, {in | out}, *hop-count)*
  - *send+* = tripla de M ou nulo, inicialmente (UID, out, 1)
  - *send–* = tripla de M ou nulo, inicialmente (UID, out, 1)
  - *status* = {*desconhecido, líder*}, inicialmente *desconhecido*
  - *phase* = inteiro, inicialmente 0

Conjunto de estados iniciais: um único estado com as inicializações dadas

Função de geração de mensagens para cada i:

envia valor corrente de *send+* **para** processo $i + 1$
envia valor corrente de *send–* **para** processo $i - 1$

Economizando Mensagens - Hirschberg e Sinclair (80)

Função de transição para cada *i*:

send+ := null; send– := null;
if mensagem_recebida de i – 1 é (v, out, h) **then**
case
v > u **and** h > 1 : send+ := (v, out, h-1)
v > u **and** h =1 : send– := (v, in, 1)
v = u: status := líder
endcase
if mensagem_recebida de i +1 é (v, out, h) **then**
case
v > u **and** h > 1 : send– := (v, out, h-1)
v > u **and** h =1 : send+ := (v, in, 1)
v = u: status := líder
endcase
if mensagem_recebida de i –1 é (v, in, 1) and v ≠ u **then**
send+ := (v, in, 1)
if mensagem_recebida de i +1 é (v, in, 1) and v ≠ u **then**
send– := (v, in, 1)
if mensagem_recebida de i +1 e de i – 1 é (v, in, 1) and v ≠ u **then**
phase := phase + 1;
send+ := (u, out, 2^{phase})
send– := (u, out, 2^{phase})

Economizando Mensagens - Hirschberg e Sinclair (80)

Complexidade de comunicação

- fase 0: 4n mensagens
- fase $L > 0$:
  - processo envia mensagem na fase L se não é vencido na fase L–1, ou seja, se é maior que todos os processos distantes 2^{L-1} em qualquer direção
  - de cada grupo de $2^{L-1}+1$ processos apenas um vence a fase L–1, ou seja, o número de processos que enviam mensagem na fase L é

$$\left\lfloor \frac{n}{2^{L-1}+1} \right\rfloor$$

  - assim, o número total de mensagens na fase $L > 0$ é

$$4 \times 2^{L} \times \left\lfloor \frac{n}{2^{L-1}+1} \right\rfloor < 8n$$

Economizando Mensagens - Hirschberg e Sinclair (80)

- número total de fases: no máximo

$$1 + \lfloor \log n \rfloor$$

Assim, número total de mensagens é

$$8n(1 + \lfloor \log n \rfloor)$$

ou seja, O(n log n).

Complexidade de tempo:

- cada fase L toma tempo 2 x 2L
- última fase toma tempo n (é incompleta)
- penúltima fase é L = ceiling(log n) – 1
- soma dos tempos é 2 x (1 + 2 + 4 + 8 + ... + $2^{\text{ceiling(log n)} - 1}$) + n
  - 2 x $2^{\text{ceiling(log n)}}$ + n, ou seja 3n (se n é potência de 2) ou 5n.

Algoritmos não baseados em comparação de UIDs

Algoritmo TimeSlice

- computação procede em fases; cada fase consiste de n rounds consecutivos. Cada fase é dedicada à circulação, por todo o anel, de uma mensagem carregando um UID particular. Mais especificamente, na fase v, que consiste das fases $(v–1)\ n\ +1$, $(v–1)\ n\ +\ 2$, ...vn, apenas a mensagem carregando UID v circula.
- se o processo i com UID igual a v existe, e o round $(v–1)\ n\ +\ 1$ é atingido antes de i receber qualquer mensagem, processo se elege como líder e envia mensagem com seu UID pelo anel.

Complexidade de mensagens

- O número total de mensagens é n.

Complexidade de tempo

- O tempo até o término é n x UID_{min}

Algoritmos não baseados em comparação de UIDs

Algoritmo VariableSpeeds

- cada processo origina uma mensagem com seu UID. Mensagens de processos diferentes caminham com velocidades diferentes. Mensagem com UID igual a v viaja com a velocidade de 1 mensagem a cada 2^v rounds (ou seja, cada processo no caminho espera 2^v rounds antes de repassar a mensagem).
- cada processo.

Complexidade de mensagens

- Quando mensagem com UID_{min} andou a volta completa, mensagem com o segundo menor UID andou no máximo meia volta. Ou seja, o número de mensagens utilizadas com UID_{min} é maior do que todas as outras mensagens somadas. Assim, o número total de mensagens é menor do que $2n$.

Complexidade de tempo

- O tempo até o término é n x 2^{UIDmin}