

# Algoritmos Distribuídos

# Algoritmos Básicos<sup>\*</sup>

Antonio Alfredo Ferreira Loureiro

`loureiro@dcc.ufmg.br`

`http://www.dcc.ufmg.br/~loureiro`

<sup>\*</sup>Este material está baseado no capítulo 4 do livro “An Introduction to Distributed Algorithms”, Valmir C. Barbosa, MIT Press, 1996.

# Propagação de Informação

# Suposições

- Topologia da rede representada por um grafo  $G$  não dirigido
- $inf$  é a informação a ser disseminada em  $G$
- $n$  é o número de nós em  $G$
- $m$  é o número de vértices em  $G$

# Propagação de informação (PI)

- Problema:
  - Difundir em  $G$  informação presente num subconjunto dos nós de  $G$
- Dois casos tratados:
  1. PI de um grupo de nós para todos os nós em  $G$ 
    - *Propagation of Information* ou, simplesmente, PI
  2. PI de um único nó  $s$  para todos os outros nós em  $G$  com o requisito que ao final da execução do algoritmo, o nó  $s$  tenha a confirmação do recebimento de *inf* por todos os outros nós
    - *Propagation of Information with Feedback* ou, simplesmente, PIF

# Estratégia para resolução do problema PI

- Suponha que  $n_1$  é o nó que tenha *inf*:
  1. Calcular uma árvore geradora em  $G$
  2. {Usar a árvore geradora para fazer a difusão de *inf*}
    - (a) Nó  $n_1$  envia *inf* em todas as arestas da árvore geradora que são incidentes a  $n_1$
    - (b) Todos os outros nós fazem o envio de *inf*, como descrito em (a), exceto na aresta onde *inf* foi recebida
- Se a árvore geradora foi computada anteriormente, um algoritmo assíncrono baseado nessa estratégia tem complexidade de tempo e mensagem de  $O(n)$
- Estratégia é similar para o caso em que dois ou mais nós possuem *inf*

# Razões que podem dificultar a aplicação dessa estratégia

1. Se a árvore geradora não estiver disponível deve ser obtida
  - Há o custo de tempo e mensagens
  - No entanto, para várias difusões de mensagens o custo fica amortizado
2. Difusão sempre ocorre nos mesmos canais (arestas)
  - Ao longo do tempo, podem haver variações na qualidade do canal (*delay, jitter*) e não há garantia da disponibilidade (confiabilidade) da mesma topologia
3. Explorar outras soluções

# Solução para o problema PI baseada em “Difusão” ou “Onda de Propagação”

- Sejam os conjuntos:
  - $N_0$ : conjunto de nós que possuem *inf* inicialmente
  - $N'_0$ : conjunto de nós que não possuem *inf* inicialmente
- Princípio:
  - Fazer difusão através de inundação (*flooding*)
- Complexidade de mensagem:
  - Tende a ser maior que usando árvore geradora

# Solução para o problema PI baseada em “Difusão” ou “Onda de Propagação”

- Idéia:
  - No início, cada nó em  $N_0$  envia *inf* para todos os seus vizinhos
  - Cada nó em  $N'_0$ , ao receber *inf* pela primeira vez, envia a mensagem para todos os seus vizinhos, incluindo a aresta de onde foi recebida
    - Um nó recebe *inf* de todos os seus vizinhos
  - A mensagem *inf* é propagada a partir dos nós em  $N_0$  através de uma onda
    - Cada nó em  $N'_0$  recebe *inf* a partir de  $N_0$ , da forma mais rápida possível, apesar de falhas que possam haver em  $G$ , que ainda geram um grafo conexo
    - Vários algoritmos distribuídos funcionam desta forma

# Algoritmo A<sub>PI</sub>

▷ **Variables:**

{ $N_0$ : conjunto de nós que possuem *inf* inicialmente}

{ $n_i$ : nó genérico do grafo  $G$ }

{ $Neig_i$ : nós vizinhos a  $n_i$ }

$reached_i = \mathbf{false}$ .

{Indica se  $n_i$  já recebeu *inf* ou não}

▷ **Input:**

$msg_i = \mathbf{nil}$ .

**Action if**  $n_i \in N_0$ :

$reached_i \leftarrow \mathbf{true}$ ;

Send *inf* to all  $n_j \in Neig_i$ .

(1)

▷ **Input:**

$msg_i = \mathbf{inf}$ .

**Action:**

**if not**  $reached_i$

**then begin**

$reached_i \leftarrow \mathbf{true}$ ;

Send *inf* to all  $n_j \in Neig_i$

**end.**

(2)

# Comentários sobre $A_{PI}$

Resultado de impossibilidade:

- Dado um nó  $n_i$  com  $reached_i = \mathbf{true}$  não é possível identificar se a cópia de  $inf$  que o nó recebe em (2) é uma resposta a mensagem enviada no:
  - (i) ponto (1), onde o nó  $n_i \in N_0$  e a mensagem foi enviada por um nó vizinho  $n_j \in N'_0$
  - (ii) ponto (2), onde o nó  $n_i \in N'_0$  e a mensagem foi enviada por um nó vizinho  $n_j \in N'_0$ou
- (iii) uma mensagem que estava em trânsito de  $n_j \in Neig_i$  para  $n_i$ .

# Problema

- O que acontece no algoritmo  $A_{PI}$  se um nó não envia  $inf$  para os nós vizinhos?

# Propagação de informação com realimentação (PIF)

- Problema:
  - Difundir em  $G$  informação presente em um único nó  $s$  para todos os outros nós em  $G$  com o requisito que ao final da execução do algoritmo, o nó  $s$  tenha a confirmação do recebimento de *inf* por todos os outros nós
- Estratégias para resolução do problema PIF:
  - Usando árvore geradora
  - Usando difusão (*flooding*)

# Solução baseada em árvore geradora

- Suposições:
  - $n_1$  é o único nó que inicialmente possui *inf* (raiz da árvore)
  - Todos os nós  $n_i, i \neq 1$ , possuem um vizinho especial,  $parent_i$ , no caminho na árvore de  $n_i$  a  $n_1$

# Solução baseada em árvore geradora

- Algoritmo:
  1. Calcular uma árvore geradora em  $G$ , sendo  $n_1$  a raiz da árvore  
{Usar a árvore geradora para fazer a difusão de  $inf$ }
  2. Nó  $n_1$  envia  $inf$  em todas as arestas da árvore geradora que são incidentes a  $n_1$
  3. Um nó  $n_i, i \neq 1$ , ao receber  $inf$  pela primeira vez de algum vizinho  $n_j$ , marca  $parent_i$ , com o vizinho  $n_j$
  4. Se  $n_i$  não é um nó folha, então  $n_i$  envia  $inf$  em todas as arestas da árvore geradora que são incidentes a  $n_i$ , exceto na aresta que leva a  $parent_i$  ( $n_j$ )
  5. Se  $n_i$  é um nó folha, então  $n_i$  envia  $inf$  imediatamente para  $parent_i$  ao receber  $inf$  pela primeira vez

# Solução baseada em árvore geradora

- Algoritmo (continuação):
  6. Um nó  $n_i, i \neq 1$ , ao receber *inf* em todas as arestas da árvore geradora que são incidentes a  $n_i$ , exceto na aresta que leva a  $parent_i (n_j)$ , envia *inf* para  $parent_i$
  7. O nó  $n_1$ , ao receber *inf* em todas as arestas da árvore geradora que são incidentes a  $n_1$ , sabe que a mensagem foi propagada e recebida por todos os nós da árvore
- Se a árvore geradora foi computada anteriormente, um algoritmo assíncrono baseado nessa estratégia tem complexidade de tempo e mensagem de  $O(n)$

# Solução para o problema PIF baseada em “Difusão”

- Idéia:
  - Similar à solução para o problema PI
  - Utiliza a variável  $parent_i$ , como descrito na solução com árvore geradora
  - Executa os passos (2) a (7) do algoritmo utilizando a árvore geradora, exceto que as mensagens são propagadas por todas as arestas, e não apenas aquelas presentes na árvore geradora

# Algoritmo *A\_PIF*

▷ **Variables:**

$parent_i = \mathbf{nil};$

{Vizinho de  $n_i$  por onde  $inf$  chegou inicialmente}

$count_i = 0;$

{Número de cópias recebidas de  $inf$ }

$reached_i = \mathbf{false}.$

{Indica se  $n_i$  já recebeu  $inf$  ou não}

▷ **Input:**

$msg_i = \mathbf{nil}.$

**Action if**  $n_i \in N_0:$

(1)

$reached_i \leftarrow \mathbf{true};$

Send  $inf$  to all  $n_j \in Neig_i.$

# Algoritmo *A\_PIF*

▷ **Input:**

$msg_i = inf \mid origin_i(msg_i) = (n_i, n_j).$

**Action:**

(2)

$count_i \leftarrow count_i + 1$

**if not**  $reached_i$

**then begin**

$reached_i \leftarrow \mathbf{true};$

$parent_i \leftarrow n_j;$

Send  $inf$  to all  $n_k \in Neig_i \mid n_k \neq parent_i.$

**end;**

**if**  $count_i = |Neig_i|$

**then if**  $parent_i \neq \mathbf{nil}$

**then** Send  $inf$  to  $parent_i.$

# Observações sobre o algoritmo $A_{PIF}$

- Coleção de variáveis  $parent_i$  para todos  $n_i \in N$  estabelece a árvore geradora com raiz em  $n_1$

- Construção da árvore envolve o seguinte número de mensagens:

$$|Neig_1| + \sum_{n_i \in N'_0} (|Neig_i| - 1) = 2m - n + 1 = O(m)$$

- Todo o algoritmo executa em  $2m = O(m)$  mensagens
- O tempo para construção da árvore e de execução do algoritmo é  $O(n)$

# Observações sobre o algoritmo $A\_PIF$

- Seja  $d$  o número de arestas no caminho mais longo na árvore entre o nó  $n_i$  e uma folha em  $T_i$
- No algoritmo  $A\_PIF$ , nó  $n_i \neq n_1$  envia  $inf$  para  $parent_i$  num tempo no máximo  $2d$  depois de ter recebido  $inf$  pela primeira vez
- No instante em que  $inf$  é enviada, cada nó em  $T_i$  recebeu  $inf$
- Nó  $n_1$  recebe  $inf$  em tempo no máximo  $O(n)$

# Múltiplas instâncias concorrentes do algoritmo de PI

- Problema relacionado ao algoritmo  $A\_PI$ :
  - Enviar uma série de mensagens  $inf_1, inf_2, \dots$
- Possível solução:
  - Ter em cada nó  $n_i$  uma variável booleana  $reached_i^k$  associada a  $inf_k$ , para  $k \geq 1$
- Problema decorrente do uso de variáveis booleanas:
  - Se o número de mensagens não é conhecido a priori, não se pode definir quantas variáveis serão necessárias
  - Deve-se inspecionar o cabeçalho para saber o identificador da mensagem
- E se as arestas em  $G$  utilizarem a política FIFO?
  - Cada nó recebe as mensagens na ordem em que foram enviadas

# Múltiplas instâncias concorrentes do algoritmo de PI

- Estratégia:
  - Cada nó  $n_i$  deve possuir  $|Neig_i|$  contadores, um para cada aresta incidente a  $n_i$ , chamados  $count_i^j$  para  $n_j \in |Neig_i|$
  - O contador  $count_i^j$  é incrementado toda vez que  $n_i$  recebe uma mensagem vinda de  $n_j$
  - O valor do contador  $count_i^j$  indica o número da última mensagem recebida em  $n_i$  vinda de  $n_j$

# Múltiplas instâncias concorrentes do algoritmo de PI

- Observações sobre o uso de um contador e variável booleana:
  - Cada contador está associado a uma aresta, e o número de arestas é finito
  - Cada variável booleana está associada a uma mensagem, que pode ser um número não conhecido a priori
- Como saber que uma mensagem  $n_l \in Neig_i$  está sendo recebida em  $n_i$  pela primeira vez?
  - Basta verificar

$$count_i^l > count_i^j$$

para todo  $n_j \in Neig_i$  tal que  $j \neq l$

# Problema

- Escreva o algoritmo que trata de múltiplas instâncias concorrentes do algoritmo *A\_PI*

# Múltiplas instâncias concorrentes do algoritmo de PIF

- Problema:
  - Nó  $n_1$  envia uma seqüência de mensagens
- Estratégia:
  - Usar variáveis booleanas, mas com os problemas já mencionados
  - Pensar em outra solução, já que com canais FIFO não é possível resolver

# Problema

- Mostre, através de um exemplo, que arestas FIFO não são suficientes para garantir que mensagens são recebidas em todos os nós na ordem enviada pelo nó  $n_1$ , no contexto de múltiplas instâncias concorrentes do algoritmo *A\_PIF*

# Conectividade em Grafos

# Introdução

- Problema:
  - Dado um grafo  $G$ , cada nó (vértice) em  $N$  deve descobrir as identificações de todos os outros nós aos quais está conectado por um caminho em  $G$
- Importância do problema:
  - Em um sistema sujeito a falhas, é importante saber os identificadores dos nós de um componente conectado
- Contexto da solução proposta;
  - Algoritmo não é adequado para o caso de  $G$  mudar dinamicamente
  - Mostra o uso da técnica de PI

# Observações

- Pode ser inicializado por qualquer nó em  $N$ 
  - Espontaneamente, se nó  $\in N_0$
  - Ao receber a primeira mensagem
- Em qualquer caso, nó  $n_i$  propaga sua identificação  $id_i$  baseado no algoritmo  $A\_PIF$
- Está sendo suposto que as arestas em  $G$  possuem a propriedade FIFO
- Basicamente, nó  $n_i$  participa em tantas instâncias concorrentes de  $A\_PIF$  quantos são os nós de  $G$ 
  - Cada instância é gerada por um nó diferente

# Algoritmo *A\_Test\_Connectivity*

▷ **Variables:**

$parent_i^k = \mathbf{nil}$  for all  $n_k \in N$ ;

$count_i^k = 0$  for all  $n_k \in N$ ;

$reached_i^k = \mathbf{false}$  for all  $n_k \in N$ ;

$initiated_i = \mathbf{false}$ .

{nó em  $Neig_i$  de onde recebeu  $id_k$ }

{# vezes que  $id_k$  foi recebido}

{indica se  $id_k$  foi recebido pelo menos uma vez}

{indica se  $n_i$  já fez iniciou a computação ou não}

▷ **Input:**

$msg_i = \mathbf{nil}$ .

**Action if**  $n_i \in N_0$ :

$initiated_i \leftarrow \mathbf{true}$ ;

$reached_i^i \leftarrow \mathbf{true}$ ;

Send  $id_i$  to all  $n_j \in Neig_i$ .

(1)

# Algoritmo A\_Test\_Connectivity

▷ **Input:**

$msg_i = id_k \mid origin_i(msg_i) = (n_i, n_j)$  for some  $n_k \in N$ .

**Action:**

(2)

**if not**  $initiated_i$  {se nó  $n_i$  recebeu uma  $\langle msg \rangle$  então não é um nó isolado}

**then begin**

$initiated_i \leftarrow \mathbf{true};$

$reached_i^i \leftarrow \mathbf{true};$

Send  $id_i$  to all  $n_l \in Neig_i$

**end;**

$count_i^k \leftarrow count_i^k + 1;$

{ $\langle msg \rangle$  recebida é proveniente de  $n_k$ }

**if not**  $reached_i^k$

**then begin**

$reached_i^k \leftarrow \mathbf{true};$

$parent_i^k \leftarrow n_j;$

Send  $id_k$  to every  $n_l \in Neig_i \mid n_l \neq parent_i^k$

**end;**

**if**  $count_i^k = |Neig_i|$

**then if**  $parent_i^k \neq \mathbf{nil}$

**then** Send  $id_k$  to  $parent_i^k$ .

# Menor Distância

# Menor distância

- Problema:
  - Determinar a menor distância em  $G$  entre todos os pares de nós
  - Distância é medida em número de arestas
- Resultado final:
  - Distância de cada nó  $n_i$  para todos os outros
  - Nó em  $Neig_i$  que está em cada caminho mais curto
    - Permite que mensagens sejam roteadas

# Algoritmo síncrono para calcular a menor distância

- Princípio:
  - O algoritmo funciona em pulsos síncronos
- Idéia básica:
  - Cada nó possui uma identificação
  - Em  $s = 0$  cada nó envia sua identificação para os seus vizinhos
  - Em  $s = i, i > 0$ , cada nó envia para os vizinhos as identificações recebidas durante  $s - 1$
  - $N_0 = N$
- O que acontece ao longo do processamento:
  - Para  $s = 1$ , cada nó conhece as identificações dos nós que estão a distância 1
  - Para  $s = i, i \geq 0$ , cada nó conhece as identificações dos nós que estão a distância  $i$

# Algoritmo *S\_Compute\_Distances*

▷ **Variables:**

$dist_i^i = 0;$  {distância de  $i$  para  $i$ }  
 $dist_i^k = n \forall n_k \in N \mid k \neq i;$  {menor distância de  $i$  para  $k$ }  
 $first_i^k = \mathbf{nil} \forall n_k \in N \mid k \neq i;$  {nó em  $Neig_i$  no caminho mais curto para  $n_k \neq n_i$ }  
 $set_i = \{id_i\}.$  {conjunto de identificações a serem enviadas para os vizinhos}

▷ **Input:**

$s = 0, MSG_i(0) = \emptyset.$

{Todos os nós estão em  $N_0$ }

**Action if**  $n_i \in N_0:$

Send  $set_i$  to all  $n_j \in Neig_i.$

(1)

# Algoritmo *S\_Compute\_Distances*

▷ **Input:**

$0 < s \leq n - 1, MSG_i(s) \mid origin_i(set_j) = (n_i, n_j)$  for  $set_j \in MSG_i(s)$

**Action:**

(2)

$set_i = \emptyset;$

**for all**  $set_j \in MSG_i(s)$  **do**

**for all**  $id_k \in set_j$  **do**

**if**  $dist_i^k > s$

**then begin**

$dist_i^k \leftarrow s;$

$first_i^k \leftarrow n_j;$

$set_i \leftarrow set_i \cup \{id_k\}.$

**end;**

Send  $set_i$  to all  $n_k \in Neig_i.$

# Comentários sobre o algoritmo *S\_Compute\_Distances*

- Cada mensagem com a identificação passa por cada aresta duas vezes
  - Logo, o número de mensagens é  $2nm = O(nm)$
  - No entanto, o comprimento de cada mensagem varia
  - Se a identificação pode ser expressa em  $\lceil \log n \rceil$  bits, então a complexidade em bits é  $O(nm \log n)$
- A complexidade em tempo é  $O(n)$ 
  - Número de pulsos para que todos os identificadores sejam propagados

# Algoritmo assíncrono para calcular a menor distância

- Estratégia:
  - Usar um algoritmo de “barreira de sincronização
- Suposições:
  - Arestas em  $G$  têm a propriedade FIFO
- Comentários:
  - Existem algoritmos mais eficientes, mas é bastante utilizado
  - É simples e ilustra técnicas de sincronização

# Algoritmo A\_Compute\_Distances

## ▷ Variables:

$dist_i^i = 0;$  {distância de  $i$  para  $i$ }

$dist_i^k = n \ \forall n_k \in N \mid k \neq i;$  {menor distância de  $i$  para  $k$ }

$first_i^k = \mathbf{nil} \ \forall n_k \in N \mid k \neq i;$  {nó em  $Neig_i$  no caminho mais curto para  $n_k \neq n_i$ }

$set_i = \{id_i\}.$  {conjunto de identificações a serem enviadas para os vizinhos}

$level_i^j = -1 \ \forall n_j \in Neig_i;$  {indica que conjuntos de identificações  $n_i$  recebeu de  $n_j$ }

{ $level_i^j = d, 0 \leq d < n$ , sse  $n_i$  recebeu de  $n_j$ , os conjuntos que estão a  $d$  arestas de  $n_j$ }

$state_i = 0;$  {indica que conjuntos de identificações  $n_i$  recebeu de seus vizinhos}

{ $state_i = d, 0 \leq d < n$ , sse  $n_i$  recebeu a identificação de todos os nós que estão a  $d$  arestas de  $n_i$ }

$initiated_i = \mathbf{false}.$  {indica se  $n_i \in N_0$ }

## ▷ Input:

$msg_i = \mathbf{nil}.$

**Action if**  $n_i \in N_0:$

$initiated_i \leftarrow \mathbf{true};$

Send  $set_i$  to all  $n_j \in Neig_i.$

(1)

# Algoritmo A\_Compute\_Distances

▷ **Input:**

$msg_i = set_j \mid origin_i(msg_i) = (n_i, n_j).$

**Action:**

(2)

**if not**  $initiated_i$

**then begin**

$initiated_i \leftarrow \mathbf{true};$  Send  $set_i$  to all  $n_j \in Neig_i.$

**end;**

**if**  $state_i < n - 1$  {se  $state_i = n - 1$  então  $n_i$  já recebeu todas as identificações em  $N$ }

**then begin**

$level_i^j \leftarrow level_i^j + 1;$  { $n_i$  está recebendo mais um conjunto de  $n_j$ }

**for all**  $id_k \in set_j$  **do**

**if**  $dist_i^k > level_i^k + 1$  {descobre o caminho mais curto de  $n_i$  para  $n_k$ }

**then begin**

$dist_i^k \leftarrow level_i^k + 1;$   $first_i^k \leftarrow n_j;$

**end;**

**if**  $state_i \leq level_i^j \forall n_j \in Neig_i$

**then begin**

$state_i \leftarrow state_i + 1;$   $set_i \leftarrow \{id_k \mid n_k \in N \wedge dist_i^k = state_i\};$

Send  $set_i$  to all  $n_k \in Neig_i.$

**end;**

**end;**