Распределенные информационно-аналитические системы

Лекция № 11.

Волновые алгоритмы распространения информации

Профессор кафедры КБ-2: д.т.н. Шатовкин Р.Р.

Учебные цели:

Изучить основы научных знаний по волновым алгоритмам: кольцевому алгоритму; древовидному алгоритму; эхо-алгоритму; алгоритму голосования; фазовому алгоритму; алгоритму Финна.

Учебные вопросы:

1. Кольцевой алгоритм.

2. Древовидный алгоритм.

3. Эхо-алгоритм.

4. Алгоритм голосования.

5. Фазовый алгоритм.

6. Алгоритм Финна.

1. Кольцевой алгоритм

Предположим, что для каждого процесса p задан сосед $Next_p$ такой, что все каналы, выбранные таким образом, составляют Гамильтонов цикл.

Алгоритм является централизованным; инициатор посылает сообщение <tok> (называемое маркером) вдоль цикла, каждый процесс передает его дальше и когда оно возвращается к инициатору, инициатор принимает решение.

Кольцевой алгоритм:

Для инициатора:

begin send <tok> to $Next_p$;

receive <tok>;

decide end

Для не-инициатора:

begin receive <tok>;

send <tok> to $Next_p$ end

Теорема 1. Кольцевой алгоритм является волновым алгоритмом.

Доказательство. Обозначим инициатор через p_0. Так как каждый процесс посылает не более одного сообщения, алгоритм передает в целом не больше N сообщений.

За ограниченное количество шагов алгоритм достигает заключительной конфигурации. В этой конфигурации p_0 уже переслал маркер, то есть выполнил оператор send в своей программе. Кроме того, ни одно сообщение <tok> не передается ни по одному каналу, иначе оно может быть получено, и конфигурация не будет заключительной. Также, ни один процесс, кроме p_0, не «задерживает» маркер (то есть получил, но не передал дальше <tok>), иначе процесс может послать <tok> и конфигурация не будет конечной. Следовательно,

(1) p_0 отправил маркер,

(2) для любого p, пославшего маркер, $Next_p$ получил маркер, и

(3) каждый $p \neq p_0$, получивший маркер, отправил маркер. Из этого и свойства Next следует, что каждый процесс отправил и получил маркер.

Так как p_0 получил маркер и конфигурация конечна, p_0 выполнил оператор decide.

Получение и отправка <tok> каждым процессом $p \neq p_0$ предшествует получению маркера процессом p_0, следовательно, условие зависимости выполнено.

2. Древовидный алгоритм

Предполагается, что алгоритм инициируют все листья дерева. Каждый процесс в алгоритме посылает ровно одно сообщение. Если процесс получил сообщение по всем инцидентным каналам, кроме одного (это условие изначально выполняется для листьев), процесс отправляет сообщение по оставшемуся каналу. Если процесс получил сообщения через все инцидентные каналы, он принимает решение.

Древовидный алгоритм:

var rec_p [q] for each q $\in$ $Neigh_p$: boolean init false ;

(* rec_p [q] = true , если p получил сообщение от q *)

begin while # {q : rec_p [q] is false } > 1 do

begin receive <tok> from q ;

rec_p [q] := true end ;

(* Теперь остался один q_0 , для которого rec_p [q_0] = false *)

send <tok> to q_0 with rec_p [q_0] is false ;

x : receive <tok> from q_0 ;

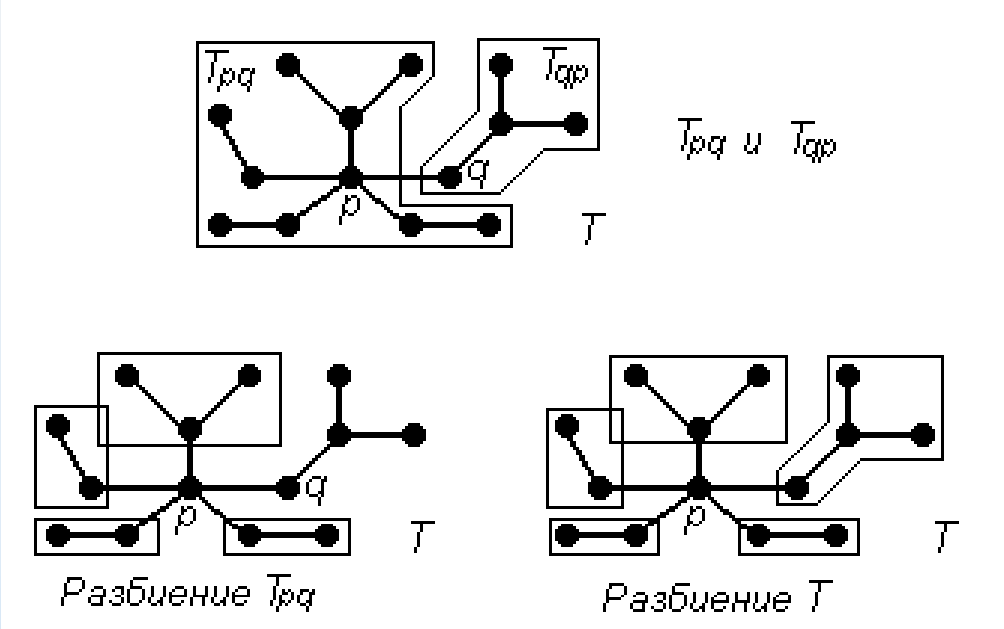
rec_p [q_0] := true ;

decide

(* Сообщить другим процессам о решении: forall q $\in$ $Neigh_p$, q $\neq$ q_0 do send <tok> to q *)

end

Чтобы показать, что этот алгоритм является волновым, введем некоторые обозначения. Пусть f_{pq} – событие, где p посылает сообщение q, а g_{pq} – событие, где q получает сообщение от p. Через T_{pq} обозначим подмножество процессов, которые достижимы из p без прохождения по дуге pq (процессы на стороне p дуги pq) (рисунок 1).



$$T_{pq} = \bigcup_{r \in Neigh_p \setminus \{q\}} T_{rp} \cup \{p\}$$

и

$$P = \{p\} \cup \bigcup_{r \in Neigh_p} T_{rp}$$

Рисунок 1 – Поддеревья T_{pq}

В древовидном алгоритме существует два процесса, которые получают сообщения через все свои каналы и принимают решение; все остальные тем временем ожидают сообщения со счетчиком команд, установленным на x, в заключительной конфигурации. Если к программе добавить оператор forall, то все процессы принимают решение и в конечной конфигурации каждый процесс находится в конечном состоянии.

Модифицированная программа использует 2N – 2 сообщений.

Теорема 2. Древовидный алгоритм является волновым алгоритмом.

Доказательство. Так как каждый процесс посылает не более одного сообщения, в целом алгоритм использует не более N сообщений. Отсюда следует, что алгоритм достигает заключительной конфигурации Υ за конечное число шагов; мы покажем, что в Υ хотя бы один процесс выполняет событие decide.

Пусть F – количество бит rec со значением false в Υ, а K – количество процессов, которые уже послали сообщения в Υ. Так как в Υ не передается ни одно сообщение (иначе Υ не была бы заключительной), $F = (2N - 2) - K$; общее число битов rec равно $2N - 2$, а K из них равны true.

Предположим, что ни один процесс в Υ не принял решения. $N - K$ процессов, которые еще не послали сообщение в Υ, содержат хотя бы по два бита rec, равных false; иначе они бы могли послать сообщение, что противоречит тому, что Υ – заключительная конфигурация. K процессов, которые послали сообщение в Υ, содержат хотя бы один бит rec, равный false; иначе они могли бы принять решение, что противоречит тому, что Υ – заключительная конфигурация. Итак, $F \geq 2(N - K) + K$, а из $(2N - 2) - K \geq 2(N - K) + K$ следует, что $-2 \geq 0$; мы пришли к противоречию, следовательно, хотя бы один процесс в Υ принимает решение.

Наконец, нужно показать, что решению предшествует событие в каждом процессе. Пусть f_{pq} – событие, где p посылает сообщение q, а g_{pq} – событие, где q получает сообщение от p. Применяя индукцию по событиям получения сообщений, можно доказать, что $\forall\ s \in T_{pq}\ \exists\ e \in C_s : e\ \pi\ g_{pq}$.

Предположим, что это выполняется для всех событий получения сообщений, предшествующих g_{pq}. Из того, что событию g_{pq} предшествует f_{pq} (в процессе p), и из алгоритма p следует, что для всех $r \in Neigh_p$ при $r \neq q$, g_{rp} предшествует f_{pq}. Из гипотезы индукции следует, что для всех таких r и для всех $s \in T_{rp}$ существует событие $e \in C_s$, где $e\ \pi\ g_{rp}$, следовательно, $e\ \pi\ g_{pq}$.

Решению d_p в p предшествуют g_{rp} для всех $r \in Neigh_p$, откуда следует, что $\forall\ s \in P\ \exists\ e \in C_s : e\ \pi\ d_p$.

3. Эхо-алгоритм

Эхо-алгоритм – это централизованный волновой алгоритм для сетей произвольной топологии. Впервые он был представлен Чангом и поэтому иногда называется эхо-алгоритмом Чанга. Более эффективная версия, которая представлена к рассмотрению в данном учебном вопросе, была предложена Сегаллом.

Алгоритм распространяет сообщения <tok> по всем процессам, таким образом определяя остовное дерево, как определено в Лемме 2 предыдущей лекции. Маркеры «отражаются» обратно через ребра этого дерева аналогично потоку сообщений в древовидном алгоритме.

Инициатор посылает сообщения всем своим соседям. После получения первого сообщения не-инициатор пересылает сообщения всем своим соседям, кроме того, от которого было получено сообщение. Когда не-инициатор получает сообщения от всех своих соседей, эхо пересылается родителю (father). Когда инициатор получает сообщения от всех своих соседей, он принимает решение.

Эхо-алгоритм:

var rec_p : integer init 0 ; (* Счетчик полученных сообщений *)
$father_p$: P init udef ;
Для инициатора:
begin forall q $\in$ $Neigh_p$ do send <tok> to q ;
while rec_p < # $Neigh_p$ do
begin receive <tok> ;
rec_p := rec_p + 1 end ;
decide
end ;
Для не-инициатора:
begin receive <tok> from neighbor q ;
$father_p$:= q ;
rec_p := rec_p + 1 ;
forall q $\in$ $Neigh_p$, q $\neq$ $father_p$ do send <tok> to q ;
while rec_p < # $Neigh_p$ do
begin receive <tok> ;
rec_p := rec_p + 1 end ;
send <tok> to $father_p$
end

Теорема 3. Эхо-алгоритм является волновым алгоритмом.

Доказательство. Так как каждый процесс посылает не более одного сообщения по каждому инцидентному каналу, количество сообщений, пересылаемых за каждое вычисление, конечно. Пусть Υ – конечная конфигурация, достигаемая в вычислении C с инициатором p_0.

Для этой конфигурации определим (подобно определению в Лемме 2 предыдущей лекции) граф $T = (P, E_T)$, где $pq \in E_T \Leftrightarrow father_p = q$. Чтобы показать, что этот граф является деревом, нужно показать, что количество ребер на единицу меньше, чем количество вершин (Лемма 2 утверждает, что T – дерево, но предполагается, что алгоритм является волновым, что нам еще нужно доказать). Отметим, что каждый процесс, участвующий в C, посылает сообщения всем своим соседям, кроме соседа, от которого он получил первое сообщение (если процесс – не-инициатор). Отсюда следует, что все его соседи получают хотя бы одно сообщение в C и также участвуют в C. Из этого следует, что $father_p \neq udef$ для всех $p \neq p_0$. Что T не содержит циклов, можно показать, как в доказательстве Леммы 2.

В корне дерева находится p_0; обозначим через T_p множество вершин в поддереве p. Ребра сети, не принадлежащие T, называются ***листовыми ребрами*** (frond edges). В Υ каждый процесс p, по крайней мере, послал сообщения всем своим соседям, кроме родителя $father_p$, следовательно, каждое листовое ребро передавало в C сообщения в обоих направлениях. Пусть f_p – событие, в котором p посылает сообщение своему родителю (если в C это происходит), а g_p – событие, в котором родитель p получает сообщение от p (если это происходит). С помощью индукции по вершинам дерева можно показать, что

(1) C содержит событие f_p для любого $p \neq p_0$;

(2) для всех $s \in T_p$ существует событие $e \in C_s$ такое, что $e \pi g_p$.

Рассмотрим следующие два случая.

p – лист. p получил в C сообщение от своего родителя и от всех других соседей (так как все остальные каналы – листовые). Таким образом, посылка <to> родителю p была возможна, и, так как Υ – конечная конфигурация, это произошло. T_p содержит только p, и, очевидно, $f_p \geq g_p$.

p – не лист. p получил в C сообщение от своего родителя и через все листовые ребра. По индукции, C содержит f_p' для каждой дочерней вершины p′ вершины p, и, так как Υ – конечная конфигурация, C также содержит g_p'. Следовательно, посылка <tok> родителю p была возможна, и, так как Υ – конечная конфигурация, это произошло. T_p состоит из объединения T_p' по всем дочерним вершинам p и из самого p. С помощью индукции можно показать, что в каждом процессе этого множества существует событие, предшествующее g_p.

Отсюда следует, также, что p_0 получил сообщение от каждого соседа и выполнил событие decide, которому предшествуют события в каждом процессе.

Остовное дерево, которое строится в вычислении эхо-алгоритма, иногда используют в последовательно выполняемых алгоритмах.

В последнем такте алгоритма каждый процесс (кроме p_0) запомнил, какой сосед в дереве является его родителем, но не запомнил дочерних вершин.

В алгоритме одинаковые сообщения принимаются от родителя, через листовые ребра и от дочерних вершин. Если требуется знание дочерних вершин в дереве, алгоритм может быть изменен, так чтобы отправлять родителю сообщения, отличные от остальных (в последней операции отправления сообщения для не-инициаторов). Дочерними вершинами процесса тогда являются те соседи, от которых были получены эти сообщения.

4. Алгоритм опроса

В алгоритме опроса инициатор запрашивает у каждого соседа ответ на сообщение и принимает решение после получения всех ответных сообщений.

Теорема 4. Алгоритм опроса является волновым алгоритмом.

Доказательство. Алгоритм пересылает по два сообщения через каждый канал, смежный с инициатором. Каждый сосед инициатора отвечает только один раз на первоначальный опрос, следовательно, инициатор получает N – 1 ответ. Этого достаточно, чтобы принять решение, следовательно, инициатор принимает решение и ему предшествует событие в каждом процессе.

Опрос может быть использован и в сети с топологией звезда, в которой инициатор находится в центре.

Алгоритм опроса:

var rec_p : integer init 0 ; (* только для инициатора *)

Для инициатора:

begin forall q $\in$ $Neigh_p$ do send <tok> to q ;

while rec_p < # $Neigh_p$ do

begin receive <tok> ;

rec_p := rec_p + 1 end ;

decide

end ;

Для не-инициатора:

begin receive <tok> from q ;

send <tok> to q end

5. Фазовый алгоритм

Рассмотрим фазовый алгоритм, который является децентрализованным алгоритмом для сетей с произвольной топологией. Алгоритм может использоваться как волновой для ориентированных сетей.

Алгоритм требует, чтобы процессам был известен диаметр сети, обозначенный в тексте алгоритма как D. Алгоритм остается корректным (хотя и менее эффективным), если процессы вместо D используют константу $D' > D$. Таким образом, для применения алгоритма необязательно точно знать диаметр сети; достаточно, если известна верхняя граница диаметра (например, N – 1). Все процессы должны использовать одну и ту же константу D'.

Пелег дополнил алгоритм таким образом, чтобы диаметр вычислялся во время выполнения, но это расширение требует уникальной идентификации.

Общий случай. Алгоритм может использоваться в ориентированных сетях произвольной топологии, где каналы могут передавать сообщения только в одном направлении. В этом случае, соседи p являются соседями по входу (процессы, которые могут посылать сообщения p) и соседями по выходу (процессы, которым p может посылать сообщения). Соседи по входу p содержатся в множестве In_p, а соседи по выходу – в множестве Out_p.

В фазовом алгоритме каждый процесс посылает ровно D сообщений каждому соседу по выходу. Только после того, как i сообщений было получено от каждого соседа по входу, (i + 1)-е сообщение посылается каждому соседу по выходу.

***Фазовый алгоритм*:**

cons D : integer = диаметр сети ;

var rec_p [q] : 0..D init 0, для каждого q $\in$ In_p ;

(* Количество сообщений, полученных от q *)

$Sent_p$: 0..D init 0 ;

(* Количество сообщений, посланных каждому соседу по выходу *)

begin if p – инициатор then

begin forall r $\in$ Out_p do send <tok> to r ;

$Sent_p$:= $Sent_p$ + 1

end ;

while min_q Rec_p [q] < D do

begin receive <tok> (от соседа q_0) ;

Rec_p [q_0] := Rec_p [q_0] + 1 ;

if minq Rec_p [q] $\geq$ $Sent_p$ and $Sent_p$ < D then

begin forall r $\in$ Out_p do send <to > to r ;

$Sent_p$:= $Sent_p$ + 1

end

end ;

decide

end

Действительно, из текста алгоритма очевидно, что через каждый канал проходит не более D сообщений (ниже показано, что через каждый канал проходит не менее D сообщений). Если существует ребро p_q, то $f_{pq}^{(i)}$ – i-е событие, в котором p передает сообщение q, а $g_{pq}^{(i)}$ – i-е событие, в котором q получает сообщение от p. Если канал между p и q удовлетворяет дисциплине FIFO, эти события соответствуют друг другу и неравенство $f_{pq}^{(i)} \geq g_{pq}^{(i)}$ выполняется. Каузальные отношения между $f_{pq}^{(i)}$ и $g_{pq}^{(i)}$ сохраняются и в случае, если канал не является FIFO, что доказывается в следующей лемме.

Лемма 1. Неравенство $f_{pq}^{(i)}$ π $g_{pq}^{(i)}$ выполняется, даже если канал не является каналом FIFO.

Доказательство. Определим m_h следующим образом: $f_{pq}^{(mh)}$ – событие отправления сообщения, соответствующее $g_{pq}^{(h)}$, то есть в своем h-м событии получения q получает m_h-е сообщение p. Из определения каузальности $f_{pq}^{(mh)}$ p $g_{pq}^{(h)}$.

Так как каждое сообщение в C получают только один раз, все m_h различны, откуда следует, что хотя бы одно из чисел m_1, ..., m_i больше или равно i.

Выберем $j \leq i$ так, чтобы $m_j \geq i$. Тогда $f_{pq}^{(i)}$ π $f_{pq}^{(mj)}$ π $g_{pq}^{(j)}$ π $g_{pq}^{(i)}$.

Теорема 5. Фазовый алгоритм является волновым алгоритмом.

Доказательство. Так как каждый процесс посылает не более D сообщений по каждому каналу, алгоритм завершается за конечное число шагов. Пусть Υ – заключительный такт вычисления C алгоритма, и предположим, что в C существует, по крайней мере, один инициатор (их может быть больше).

Чтобы продемонстрировать, что в Υ каждый процесс принял решение, покажем сначала, что каждый процесс хотя бы один раз послал сообщения. Так как в Υ по каналам не передается ни одно сообщение, для каждого канала qp $Rec_p[q] = Sent_{pq}$. Также, так как каждый процесс посылает сообщения, как только получит сообщение сам, $Rec_p[q] > 0 \Rightarrow Sent_p > 0$. Из предположения, что существует хотя бы один инициатор p_0, для которого $Sent_{p0} > 0$, следует, что $Sent_p > 0$ для каждого p.

Впоследствии будет показано, что каждый процесс принял решение. Пусть p – процесс с минимальным значением переменной Sent в Υ, то есть для всех q $Sent_q \geq Sent_p$ в Υ. В частности, это выполняется, если q – сосед по входу p, и из $Rec_p[q] = Sent_q$ следует, что $\min_q Rec_p[q] \geq Sent_p$. Но отсюда следует, что $Sent_p = D$; иначе p послал бы дополнительные сообщения, когда он получил последнее сообщение. Следовательно, $Sent_p = D$ для всех p, и $Rec_p[q] = D$ для всех qp, откуда действительно следует, что каждый процесс принял решение.

Остается показать, что каждому решению предшествует событие в каждом процессе. Если $P = p_0, p_1, ..., p_\lambda$ ($\lambda \leq D$) – маршрут в сети, тогда, по Лемме 1:

для $0 \leq i < \lambda$ и, по алгоритму,

для $0 \leq i < \lambda - 1$.

Следовательно, так как диаметр сети равен D, для любых q и p существует маршрут $q = p_0, p_1, ..., p_\lambda = p$ длины не более D. Таким образом, для любого q существует $\lambda \leq D$ и сосед по входу r процесса p, такие, что на основании алгоритма, предшествуют d_p.

Алгоритм пересылает D сообщений через каждый канал, что приводит в сложности сообщений, равной $|E|*D$. Однако нужно заметить, что |E| обозначает количество направленных каналов. Если алгоритм используется для неориентированной сети, каждый канал считается за два направленных канала, и сложность сообщений равна $2|E|*D$.

Фазовый алгоритм для клики. Если сеть имеет топологию клика, ее диаметр равен 1; в этом случае от каждого соседа должно быть получено ровно одно сообщение, и для каждого процесса достаточно посчитать общее количество полученных сообщений вместо того, чтобы считать сообщения от каждого соседа по входу отдельно; см. Алгоритм 6.8. Сложность сообщений в этом случае равна N(N-1) и алгоритм использует только O(log N) бит оперативной памяти.

***Фазовый алгоритм для клики*:**

```
var rec_p : 0..N – 1 init 0 ;
(* Количество полученных сообщений *)
Sent_p : 0..1 init 0 ;
(* Количество сообщений, посланных каждому соседу *)
begin if p – инициатор then
begin forall r ∈ Neigh_p do send <tok> to r ;
Sent_p := Sent_p + 1
end ;
while Rec_p < # Neigh_p do
begin receive <tok> ;
Rec_p := Rec_p + 1 ;
if Sent_p = 0 then
begin forall r ∈ Neigh_p do send <tok> to r ;
Sent_p := Sent_p + 1
end
end ;
decide
end
```

6. Алгоритм Финна

Алгоритм Финна – еще один волновой алгоритм, который можно использовать в ориентированных сетях произвольной топологии. Он не требует того, чтобы диаметр сети был известен заранее, но подразумевает наличие уникальных идентификаторов процессов. В сообщениях передаются множества идентификаторов процессов, что приводит к довольно высокой битовой сложности алгоритма.

Процесс p содержит два множества идентификаторов процессов, Inc_p и $NInc_p$. Неформально говоря, Inc_p – это множество процессов q таких, что событие в q предшествует последнему произошедшему событию в p, а $NInc_p$ – множество процессов q таких, что для всех соседей r процесса q событие в r предшествует последнему произошедшему событию в p. Эта зависимость поддерживается следующим образом. Изначально $Inc_p = \{p\}$, а $NInc_p = \varnothing$ Каждый раз, когда одно из множеств пополняется, процесс p посылает сообщение, включая в него Inc_p и $NInc_p$. Когда p получает сообщение, включающее множества Inc и NInc, полученные идентификаторы включаются в версии этих множеств в процессе p. Когда p получит сообщения от всех соседей по входу, p включается в $NInc_p$. Когда два множества становятся равны, p принимает решение. Из неформального смысла двух множеств следует, что для каждого процесса q такого, что событие в q предшествует d_p, выполняется следующее: для каждого соседа r процесса q событие в r также предшествует d_p, откуда следует зависимость алгоритма.

В доказательстве корректности демонстрируется, что это выполняется для каждого p, и что из равенства двух множеств следует, что решению предшествует событие в каждом процессе.

Теорема 6. Алгоритм Финна является волновым алгоритмом.

Доказательство. Заметим, что два множества, поддерживаемые каждым процессом, могут только расширяться. Так как размер двух множеств в сумме составляет не менее 1 в первом сообщении, посылаемом по каждому каналу, и не более 2N в последнем сообщении, то общее количество сообщений ограничено 2N∗|E|.

Пусть C – вычисление, в котором существует хотя бы один инициатор, и пусть Υ – заключительный такт. Можно показать, как и в доказательстве Теоремы 5, что если процесс p отправил сообщения хотя бы один раз (каждому соседу), а q – сосед p по выходу, то q тоже отправил сообщения хотя бы один раз. Отсюда следует, что каждый процесс переслал хотя бы одно сообщение (через каждый канал).

Алгоритм Финна:

var Inc_p : set of processes init {p} ;
$NInc_p$: set of processes init ∅;
rec_p [q] : boolean for q ∈ Inp init false ;
(* признак того, получил ли p сообщение от q *)
begin if p – инициатор then
forall r ∈ Out_p do send <sets , Inc_p , $NInc_p$ > to r ;
while $Inc_p \neq NInc_p$ do
begin receive <sets , Inc, Ninc> from q_0 ;
$Inc_p := Inc_p \cup Inc$; $NInc_p := NInc_p \cup NInc$;
recp [q_0] := true ;
if ∀q ∈ In_p : rec_p [q] then $NInc_p := NInc_p \cup \{p\}$;
if Inc_p или $NInc_p$ изменились then
forall r ∈ Out_p do send <sets , Inc_p , $NInc_p$> to r
end ;
decide
end

Покажем, что в Υ каждый процесс принял решение.

Во-первых, если существует ребро pq, то $Inc_p \subseteq Inc_q$ в Υ. Действительно, после последнего изменения Inc_p процесс p посылает сообщение <sets , Inc_p, $NInc_p$>, и после его получения в q выполняется $Inc_q := Inc_q \cup Inc_p$. Из сильной связности сети следует, что $Inc_p = Inc_q$ для всех p и q. Так как выполняется $p \in Inc_p$ и каждое множество Inc содержит только идентификаторы процессов, для каждого p $Inc_p = P$.

Во-вторых, подобным же образом может быть показано, что $NInc_p = Ninc_q$ для любых p и q. Так как каждый процесс отправил хотя бы одно сообщение по каждому каналу, для каждого процесса p выполняется: $\forall q \in In_p$: recp [q], и следовательно, для каждого p выполняется: $p \in NInc_p$. Множества NInc содержат только идентификаторы процессов, откуда следует, что $NInc_p = P$ для каждого p. Из $Inc_p = P$ и $NInc_p = P$ следует, что $Inc_p = NInc_p$, следовательно, каждый процесс p в Υ принял решение.

Теперь нужно показать, что решению d_p в процессе p предшествуют события в каждом процессе. Для события e в процессе p обозначим через $Inc^{(e)}$ (или, соответственно, $NInc^{(e)}$) значение Inc_p ($NInc_p$) сразу после выполнения e. Следующие два утверждения формализуют неформальные описания множеств в начале этого раздела.

Утверждение 1. Если существует событие $e \in C_q : e \pi f$, то $q \in Inc^{(f)}$.

Доказательство. Можно показать, что $e \geq f \Rightarrow Inc^{(e)} \subseteq Inc^{(f)}$, а при $e \in C_q \Rightarrow q \in Inc^{(e)}$, что и требовалось доказать.

Утверждение 2. Если $q \in NInc^{(f)}$, тогда для всех $r \in In_q$ существует событие $e \in C_r : e \pi f$.

Доказательство. Пусть a_q – внутреннее событие q, в котором впервые в q выполняется присваивание $NInc_q := NIncq \cup \{q\}$. Событие a_q – единственное событие с $q \in NInc^{(aq)}$, которому не предшествует никакое другое событие a', удовлетворяющее условию $q \in NInc^{(a')}$; таким образом, $q \in NInc^{(f)} \Rightarrow a_q \pi f$.

Из алгоритма следует, что для любого $r \in In_q$ существует событие $e \in C_r$, предшествующее a_q. Отсюда следует результат.

Процесс p принимает решение только когда $Inc_p = NInc_p$; можно записать, что $Inc^{(dp)} = NInc^{(dp)}$. В этом случае

(1) $p \in Inc^{(dp)}$; и

(2) из $q \in Inc^{(dp)}$ следует, что $q \in NInc^{(dp)}$, откуда следует, что $In_q \subseteq Inc^{(dp)}$.

Из сильной связности сети следует требуемый результат: $Inc^{(dp)} = P$.

Выводы

В ходе лекции рассмотрены следующие вопросы:

- кольцевой алгоритм;

- древовидный алгоритм;

- эхо-алгоритм;

- алгоритм голосования;

- фазовый алгоритм;

- алгоритм Финна.

Задание на самостоятельную работу

1. Конспект лекций.

Вид и тема следующего занятия

Практическое занятие №11. Статические файлы. Логгирование