Распределенные алгоритмы

ЛЕКТОР: В.А. Захаров

Лекция 7.

Волновые алгоритмы.
Фазовый алгоритм.
Алгоритм Финна.
Распределенные алгоритмы обхода:
основные свойства.
Алгоритм обхода Тарри.
Распределенный обход в глубину.
Алгоритмы Авербаха и Сидона.

Волновые алгоритмы

Определение.

Волновым алгоритмом называется распределенный алгоритм, который удовлетворяет следующим трем требованиям.

1. **Завершение.** Каждое вычисление конечно:

$$\forall C : |C| < \infty.$$

2. **Решение.** Каждое вычисление содержит хотя бы одно событие решения:

$$\forall C : \exists e \in C : e \text{ является событием решения } \mathit{decide}.$$

3. **Зависимость.** В любом вычислении всякому событию решения предшествует в причинно-следственном отношении хотя бы одно событие в каждом из процессов:

Волновые алгоритмы

Волновые алгоритмы пригодны для решения следующих задач:

1. **Широковещательное распространение информации с подтверждением.**
2. **Синхронизации процессов системы.**
3. **Вычисления глобальной функции точной нижней грани.**
4. и многих других (избрания лидера, обнаружения завершения и др.)

Волновые алгоритмы

Основные волновые алгоритмы:

1. **Волновые алгоритмы в кольцах.**
2. **Древесный волновой алгоритм.**
3. **Алгоритм эха.**

Волновые алгоритмы

Основные волновые алгоритмы:

1. **Волновые алгоритмы в кольцах.**
2. **Древесный волновой алгоритм.**
3. **Алгоритм эха.**

Но эти алгоритмы работают только в сетях с двусторонними каналами связи (неориентированными графами).

Рассмотрим еще несколько более общих алгоритмов: фазовый алгоритм и алгоритм Финна.

Фазовый алгоритм

Фазовый алгоритм является децентрализованным алгоритмом, пригодным для сетей с произвольной топологией. Его можно использовать в качестве волнового алгоритма для ориентированных сетей.

Фазовый алгоритм

Фазовый алгоритм является децентрализованным алгоритмом, пригодным для сетей с произвольной топологией. Его можно использовать в качестве волнового алгоритма для ориентированных сетей.

В этом алгоритме требуется, чтобы все процессы располагали сведениями о диаметре сети D . Алгоритм будет оставаться корректным (хотя и менее эффективным), если все процессы будут использовать вместо D константу D' , превышающую диаметр сети.

Фазовый алгоритм

Фазовый алгоритм является децентрализованным алгоритмом, пригодным для сетей с произвольной топологией. Его можно использовать в качестве волнового алгоритма для ориентированных сетей.

В этом алгоритме требуется, чтобы все процессы располагали сведениями о диаметре сети D . Алгоритм будет оставаться корректным (хотя и менее эффективным), если все процессы будут использовать вместо D константу D' , превышающую диаметр сети.

Фазовый алгоритм можно применять для всякой ориентированной сети, по каналам которой осуществляется односторонняя передача сообщений. В этом случае соседями вершины p будут соседи на входе (процессы, которые могут отправлять сообщения процессу p) и соседи на выходе (процессы, которым p может отправлять сообщения). Соседи p на входе образуют множество In_p , а соседи на выходе — множество Out_p .

Фазовый алгоритм

Основная идея.

В фазовом алгоритме всякий процесс отправляет в точности D сообщений каждому соседу на выходе.

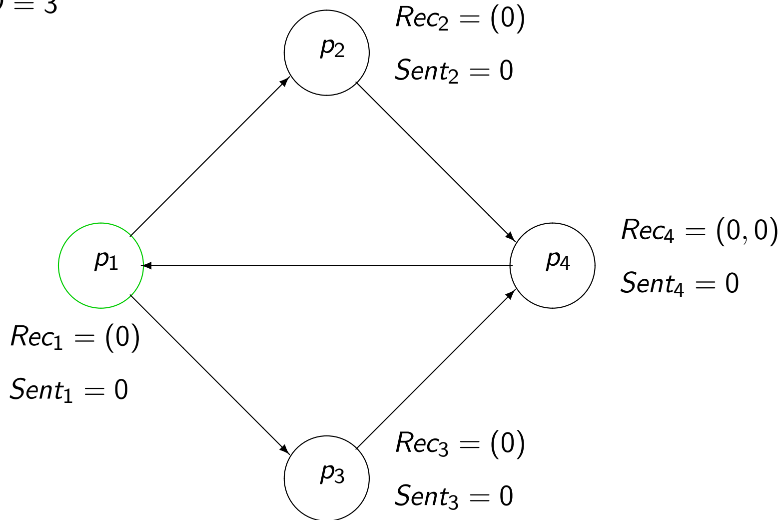
При этом $(i+1)$ -е сообщение отправляется каждому соседу на выходе только после того, как были получены i сообщений от каждого соседа на входе.

Как только от каждого соседа будет получено в точности D сообщений, процесс завершает алгоритм и принимает решение.

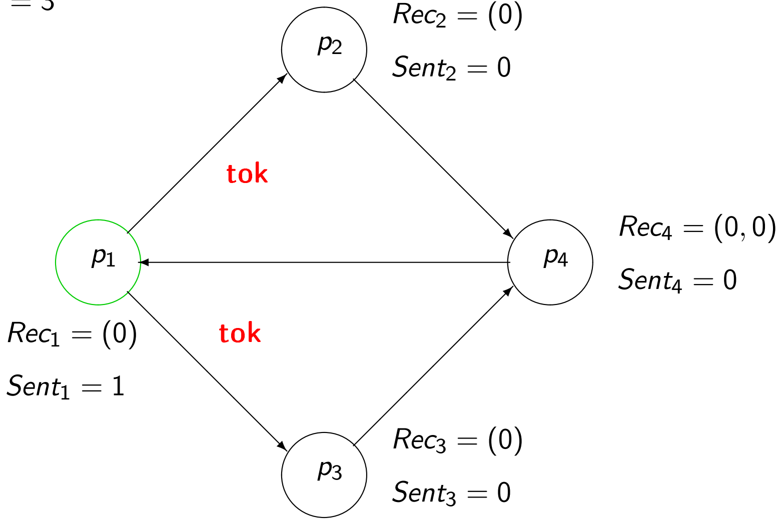
Фазовый алгоритм

```
cons D          : integer     = диаметр сети ;
var  Rec_p[q]   : 0..D        init 0 для каждого q ∈ In_p ;
     (* Число сообщений, полученных от q *)
     Sent_p     : 0..D        init 0 ;
     (* Число сообщений, отправленных каждому соседу на выходе *
begin if p is initiator then
        begin forall r ∈ Out_p do send ⟨tok⟩ to r ;
                Sent_p := Sent_p + 1 end;
      while min_q Rec_p[q] < D do
            begin receive ⟨tok⟩ (from neighbor q_0) ;
                  Rec_p[q_0] := Rec_p[q_0] + 1 ;
                  if min_q Rec_p[q] ≥ Sent_p and Sent_p < D then
                     begin forall r ∈ Out_p do send ⟨tok⟩ to r ;
                           Sent_p := Sent_p + 1 end
            end;
      decide
end
```

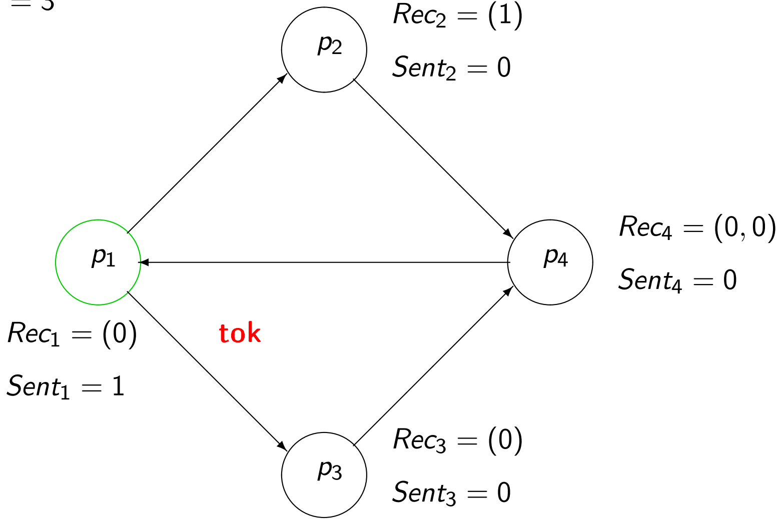
Фазовый алгоритм



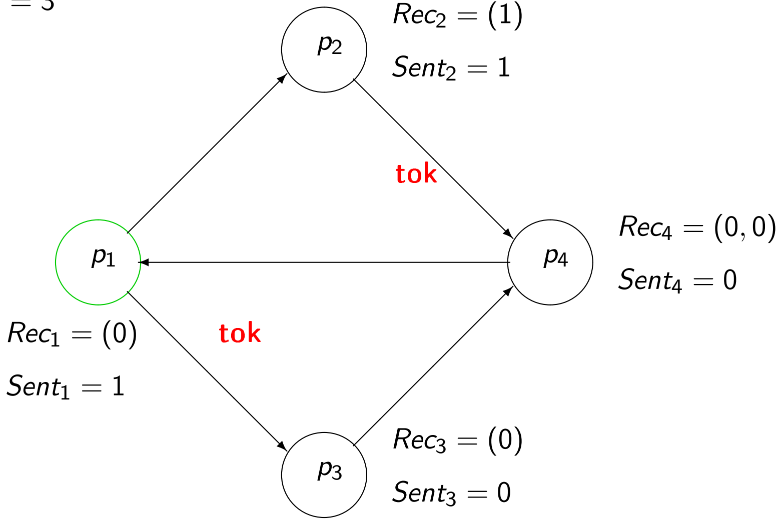
Фазовый алгоритм



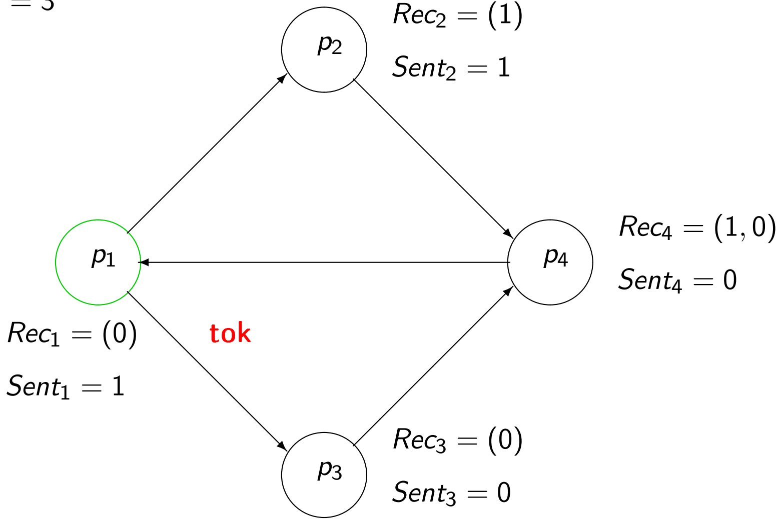
Фазовый алгоритм



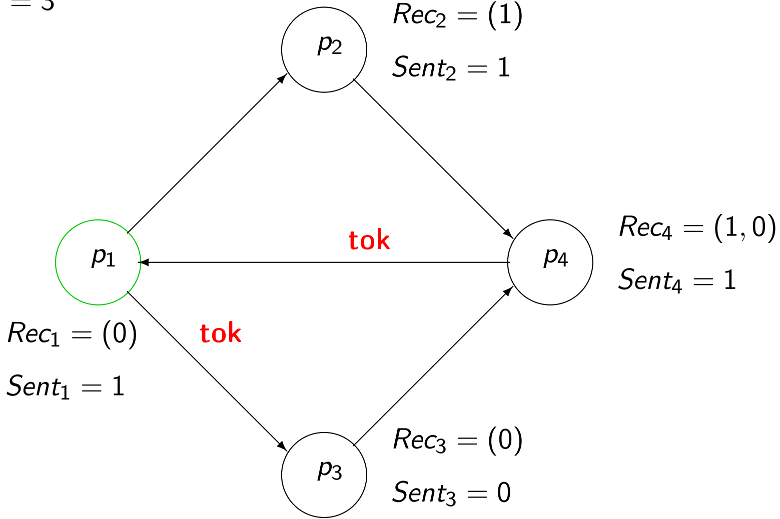
Фазовый алгоритм



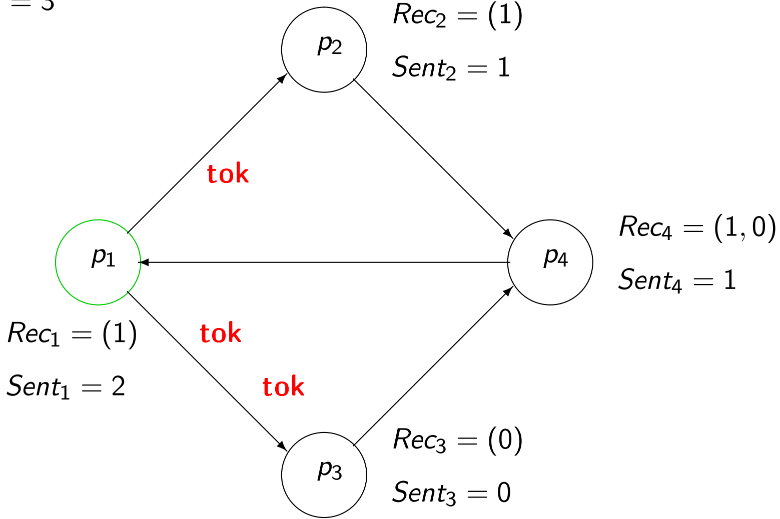
Фазовый алгоритм



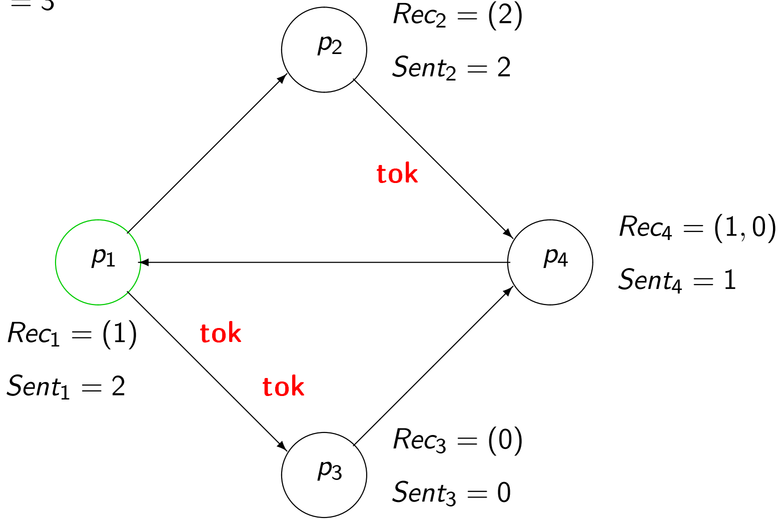
Фазовый алгоритм



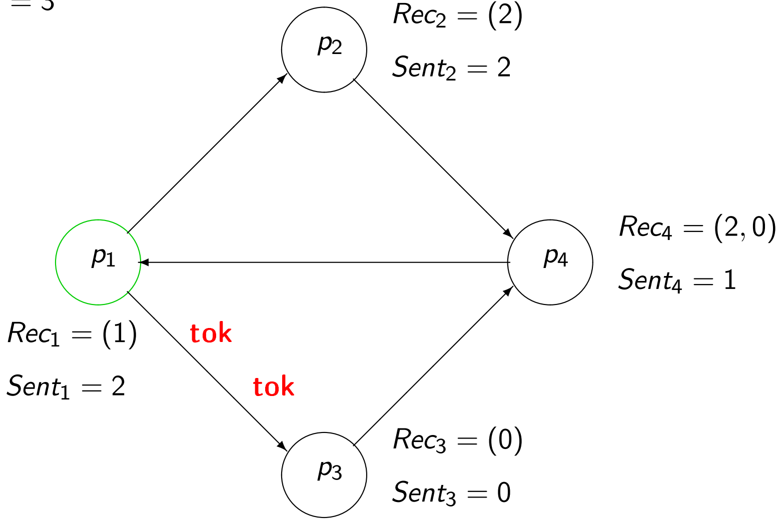
Фазовый алгоритм



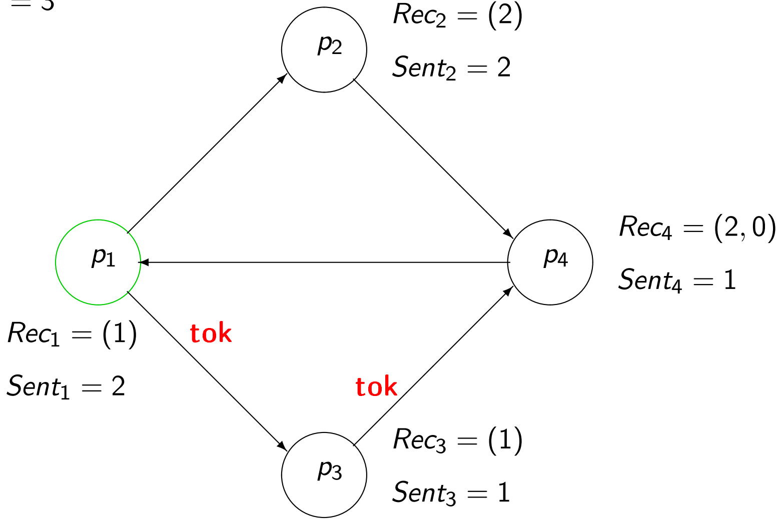
Фазовый алгоритм



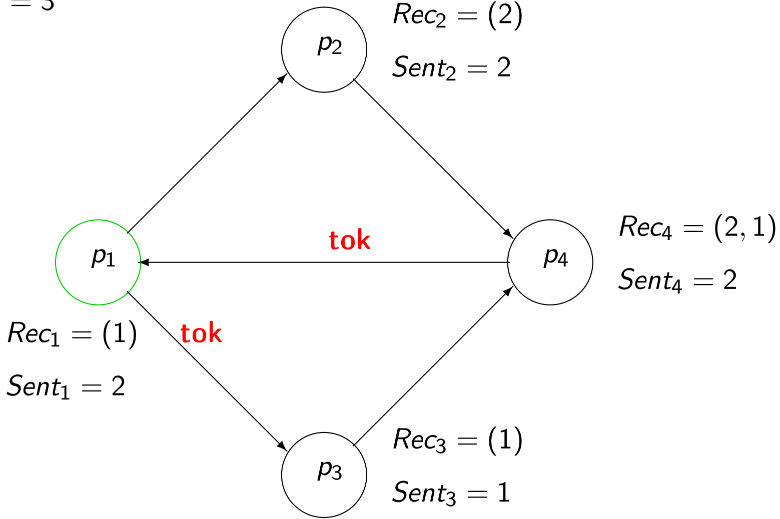
Фазовый алгоритм



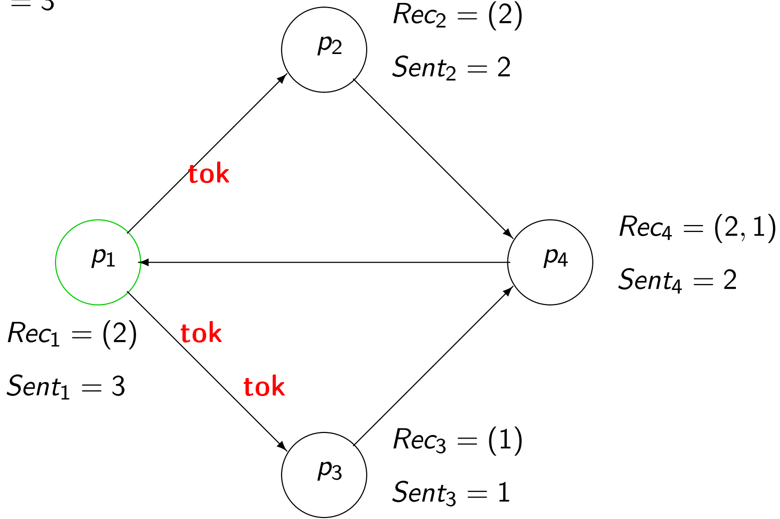
Фазовый алгоритм



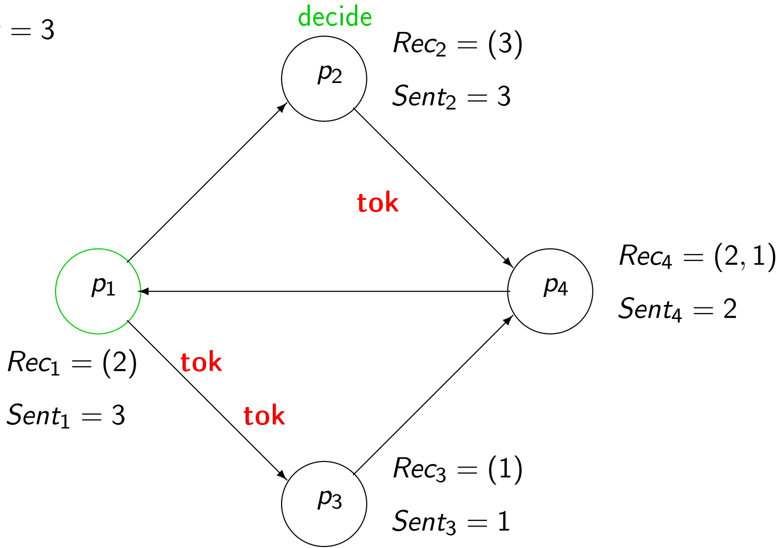
Фазовый алгоритм



Фазовый алгоритм



Фазовый алгоритм



Фазовый алгоритм

По каждому каналу передается не более D сообщений.

Для каждого ребра pq выражение

$f_{pq}^{(i)}$ обозначает i -е событие отправления процессом p сообщения процессу q ,

$g_{pq}^{(i)}$ обозначает i -е событие приема процессом q сообщения от процесса p .

Если в каналах поддерживается очередность сообщений, то $f_{pq}^{(i)} \preceq g_{pq}^{(i)}$ очевидно выполняется.

Однако отношение $f_{pq}^{(i)} \preceq g_{pq}^{(i)}$ соблюдается и в тех случаях, когда канал не является очередью.

Фазовый алгоритм

Лемма 7.1.

Соотношение $f_{pq}^{(i)} \preceq g_{pq}^{(i)}$ соблюдается и тогда, когда канал не является очередью.

Доказательство.

Пусть m_ℓ таково, что $f_{pq}^{(m_\ell)}$ — это событие отправления сообщения, которое соответствует $g_{pq}^{(\ell)}$, т.е. при выполнении ℓ -го события приема процесс q получает m_ℓ -е сообщение от p .

По определению отношения $\preceq$ имеем $f_{pq}^{(m_\ell)} \preceq g_{pq}^{(\ell)}$.

Фазовый алгоритм

Лемма 7.1.

Соотношение $f_{pq}^{(i)} \preceq g_{pq}^{(i)}$ соблюдается и тогда, когда канал не является очередью.

Доказательство.

Пусть m_ℓ таково, что $f_{pq}^{(m_\ell)}$ — это событие отправления сообщения, которое соответствует $g_{pq}^{(\ell)}$, т.е. при выполнении ℓ -го события приема процесс q получает m_ℓ -е сообщение от p.

По определению отношения $\preceq$ имеем $f_{pq}^{(m_\ell)} \preceq g_{pq}^{(\ell)}$.

Каждое сообщение принимается однократно, поэтому все m_ℓ различны. Следовательно, хотя бы одно из чисел $m_1, \ldots, m_i$ будет не меньше, чем i.

Фазовый алгоритм

Лемма 7.1.

Соотношение $f_{pq}^{(i)} \preceq g_{pq}^{(i)}$ соблюдается и тогда, когда канал не является очередью.

Доказательство.

Пусть m_ℓ таково, что $f_{pq}^{(m_\ell)}$ — это событие отправления сообщения, которое соответствует $g_{pq}^{(\ell)}$, т.е. при выполнении ℓ -го события приема процесс q получает m_ℓ -е сообщение от p .

По определению отношения $\preceq$ имеем $f_{pq}^{(m_\ell)} \preceq g_{pq}^{(\ell)}$.

Каждое сообщение принимается однократно, поэтому все m_ℓ различны. Следовательно, хотя бы одно из чисел $m_1, \ldots, m_i$ будет не меньше, чем i .

Выберем $j \leq i$ такое, что $m_j \geq i$.

Тогда $f_{pq}^{(i)} \preceq f_{pq}^{(m_j)} \preceq g_{pq}^{(j)} \preceq g_{pq}^{(i)}$.

Фазовый алгоритм

Теорема о фазовом алгоритме

Фазовый алгоритм — это волновой алгоритм.

Доказательство.

По каждому каналу отправляется не более D сообщений. Значит, все вычисления алгоритма завершаются.

Пусть γ — заключительная конфигурация вычисления C .

Фазовый алгоритм

Теорема о фазовом алгоритме

Фазовый алгоритм — это волновой алгоритм.

Доказательство.

По каждому каналу отправляется не более D сообщений. Значит, все вычисления алгоритма завершаются.

Пусть γ — заключительная конфигурация вычисления C .

Сначала покажем, что каждый процесс отправил хоть одно сообщение.

Фазовый алгоритм

Теорема о фазовом алгоритме

Фазовый алгоритм — это волновой алгоритм.

Доказательство.

По каждому каналу отправляется не более D сообщений. Значит, все вычисления алгоритма завершаются.

Пусть γ — заключительная конфигурация вычисления C .

Сначала покажем, что каждый процесс отправил хоть одно сообщение.

Т.к. в конфигурации γ в каналах нет сообщений, $Rec_p[q] = Sent_q$ для каждого канала qp .

Фазовый алгоритм

```
begin if  p is initiator then
      begin  forall r ∈ Out_p do send ⟨tok⟩ to r ;
             Sent_p := Sent_p + 1 end;
      while min_q Rec_p[q] < D do
            begin receive ⟨tok⟩ (from neighbor q_0) ;
                  Rec_p[q_0] := Rec_p[q_0] + 1 ;
                  if min_q Rec_p[q] ≥ Sent_p and Sent_p < D then
                     begin forall r ∈ Out_p do send ⟨tok⟩ to r ;
                           Sent_p := Sent_p + 1 end
            end;
      decide
end
```

Каждый инициатор отправил сообщение.

Фазовый алгоритм

begin if p is initiator **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**⟩ (from neighbor q_0) ;
      $Rec_p[q_0] := Rec_p[q_0] + 1$;
      if $\min_q Rec_p[q] \geq Sent_p$ **and** $Sent_p < D$ **then**
        begin forall $r \in Out_p$ **do** send ⟨**tok**⟩ to r ;
          $Sent_p := Sent_p + 1$ **end**
    end;
  decide
end

Каждый не-инициатор отправляет первое сообщение всем соседям сразу после получения первого сообщения от какого-нибудь соседа.

Фазовый алгоритм

Теорема о фазовом алгоритме

Фазовый алгоритм — это волновой алгоритм.

Доказательство.

По каждому каналу отправляется не более D сообщений. Значит, все вычисления алгоритма завершаются.

Пусть γ — заключительная конфигурация вычисления C .

Сначала покажем, что каждый процесс отправил хоть одно сообщение.

Т.к. в конфигурации γ в каналах нет сообщений, $Rec_p[q] = Sent_q$ для каждого канала qp .

Каждый инициатор отправил сообщение.

Каждый не-инициатор отправляет первое сообщение всем соседям сразу после получения первого сообщения.

Значит, если сильно связная сеть имеет хоть одного инициатора, то $Sent_p > 0$ для каждого узла p .

Фазовый алгоритм

Доказательство.

Покажем, что каждый процесс принял решение.

Допустим, что у процесса p в конфигурации γ переменная $Sent$ имеет наименьшее значение, т.е. $Sent_q \geq Sent_p$ для всех q .
В частности, это верно для всех процессов q , являющихся соседями p на входе.

Фазовый алгоритм

Доказательство.

Покажем, что каждый процесс принял решение.

Допустим, что у процесса p в конфигурации γ переменная $Sent$ имеет наименьшее значение, т.е. $Sent_q \geq Sent_p$ для всех q .
В частности, это верно для всех процессов q , являющихся соседями p на входе.

Тогда из равенства $Rec_p[q] = Sent_q$ следует
$\min_q Rec_p[q] \geq Sent_p$.

Фазовый алгоритм

begin if p is initiator **then**
  begin forall $r \in Out_p$ **do** send $\langle$**tok**$\rangle$ to r ;
    $Sent_p := Sent_p + 1$ **end**;
  while $\min_q Rec_p[q] < D$ **do**
    begin receive $\langle$**tok**$\rangle$ (from neighbor q_0) ;
      $Rec_p[q_0] := Rec_p[q_0] + 1$;
      if $\min_q Rec_p[q] \geq Sent_p$ **and** $Sent_p < D$ **then**
        begin forall $r \in Out_p$ **do** send $\langle$**tok**$\rangle$ to r ;
          $Sent_p := Sent_p + 1$ **end**
    end;
  decide
end

Значит, $Sent_p = D$, т.к. в противном случае p пришлось бы отправлять дополнительное сообщение, как только он в последний раз получил некоторое сообщение.

Фазовый алгоритм

Доказательство.

Покажем, что каждый процесс принял решение.

Допустим, что у процесса p в конфигурации γ переменная $Sent$ имеет наименьшее значение, т.е. $Sent_q \geq Sent_p$ для всех q .
В частности, это верно для всех процессов q , являющихся соседями p на входе.

Тогда из равенства $Rec_p[q] = Sent_q$ следует
$\min_q Rec_p[q] \geq Sent_p$.

Фазовый алгоритм

Доказательство.

Покажем, что каждый процесс принял решение.

Допустим, что у процесса p в конфигурации γ переменная $Sent$ имеет наименьшее значение, т.е. $Sent_q \geq Sent_p$ для всех q .
В частности, это верно для всех процессов q , являющихся соседями p на входе.

Тогда из равенства $Rec_p[q] = Sent_q$ следует
$\min_q Rec_p[q] \geq Sent_p$.

Значит, $Sent_p = D$; в противном случае p пришлось бы отправлять дополнительное сообщение, как только он в последний раз получил некоторое сообщение.

Значит, $Sent_p = D$ для всех p , и, следовательно, для всех каналов qp имеет место равенство $Rec_p[q] = D$.

Отсюда заключаем, что каждый процесс принял решение.

Фазовый алгоритм

Доказательство.

Покажем, что всякому решению предшествует хоть одно событие в каждом из процессов.

Фазовый алгоритм

Доказательство.

Покажем, что всякому решению предшествует хоть одно событие в каждом из процессов.

Рассмотрим путь в сети $P = p_0, p_1, \ldots, p_\ell$, где $\ell \leq D$.
По лемме 7.1. $f^{(i+1)}_{p_i p_{i+1}} \preceq g^{(i+1)}_{p_i p_{i+1}}$ верно для любого $i,\ 0 \leq i < \ell$, и, согласно алгоритму, $g^{(i+1)}_{p_i p_{i+1}} \preceq f^{(i+2)}_{p_{i+1} p_{i+2}}$ верно для любого $i,\ 0 \leq i < \ell - 1$. Следовательно, $f^{(1)}_{p_0 p_1} \preceq g^{(\ell)}_{p_{\ell-1} p_\ell}$.

Фазовый алгоритм

Доказательство.

Покажем, что всякому решению предшествует хоть одно событие в каждом из процессов.

Рассмотрим путь в сети $P = p_0, p_1, \ldots, p_\ell$, где $\ell \leq D$.
По лемме 7.1. $f^{(i+1)}_{p_i p_{i+1}} \preceq g^{(i+1)}_{p_i p_{i+1}}$ верно для любого $i,\ 0 \leq i < \ell$, и, согласно алгоритму, $g^{(i+1)}_{p_i p_{i+1}} \preceq f^{(i+2)}_{p_{i+1} p_{i+2}}$ верно для любого $i,\ 0 \leq i < \ell - 1$. Следовательно, $f^{(1)}_{p_0 p_1} \preceq g^{(\ell)}_{p_{\ell-1} p_\ell}$.

Так как диаметр сети равен D , для каждой пары процессов q и p существует путь $q = p_0, p_1, \ldots, p_\ell = p$ длины не больше D .

Фазовый алгоритм

Доказательство.

Покажем, что всякому решению предшествует хоть одно событие в каждом из процессов.

Рассмотрим путь в сети $P = p_0, p_1, \ldots, p_\ell$, где $\ell \le D$.
По лемме 7.1. $f^{(i+1)}_{p_i p_{i+1}} \preceq g^{(i+1)}_{p_i p_{i+1}}$ верно для любого $i,\ 0 \le i < \ell$, и, согласно алгоритму, $g^{(i+1)}_{p_i p_{i+1}} \preceq f^{(i+2)}_{p_{i+1} p_{i+2}}$ верно для любого $i,\ 0 \le i < \ell - 1$. Следовательно, $f^{(1)}_{p_0 p_1} \preceq g^{(\ell)}_{p_{\ell-1} p_\ell}$.

Так как диаметр сети равен D , для каждой пары процессов q и p существует путь $q = p_0, p_1, \ldots, p_\ell = p$ длины не больше D .

Поэтому для каждого q найдется такое $\ell \le D$ и такой сосед r на входе процесса p , для которых $f^{(1)}_{qq'} \preceq g^{(\ell)}_{rp}$.

А в силу устройства алгоритма событие $g^{(\ell)}_{rp}$ предшествует событию $decide_p$. □

Фазовый алгоритм

Сложность фазового алгоритма.

В алгоритме по каждому каналу передается D сообщений, и поэтому его сложность по числу обменов сообщениями равна $|E|D$.

Нужно иметь в виду, однако, что здесь $|E|$ обозначает количество greenодносторонних каналов. Если фазовый алгоритм используется для неориентированной сети, то каждый ее канал следует рассматривать как два односторонних канала, и поэтому сложность по числу сообщений будет равна $2|E|D$.

Фазовый алгоритм

Задачи.

1. Покажите, что взаимосвязь, которая проявляется в лемме 7.1., сохраняется и в том случае, когда сообщения могут утеряны в канале pq, но не сохраняется, когда сообщения могут дублироваться. Какой из этапов доказательства утратит силу, если сообщения могут дублироваться?
2. Преобразуйте фазовый алгоритм для вычисления максимума на множестве целочисленных входных данных всех процессов.
   Какова сложность по числу обменов сообщениями построенного алгоритма?
   Можно ли при помощи фазового алгоритма вычислять суммы входных данных всех процессов?

Алгоритм Финна

Алгоритм Финна — это еще один волновой алгоритм, который можно использовать для произвольных ориентированных сетей.

Здесь не требуется, чтобы диаметр сети был известен заранее, но зато этот алгоритм опирается на однозначную идентифицируемость процессов.

В сообщениях процессы обмениваются отличительными признаками, и это приводит к тому, что битовая сложность алгоритма становится достаточно большой.

Алгоритм Финна

Основная идея.

Процесс p формирует два множества отличительных признаков Inc_p и $NInc_p$.

- Inc_p — это множество таких процессов q , что некоторое событие в q предшествует (по отношению $\preceq$) самому последнему событию, случившемуся в p ,
- $NInc_p$ — множество таких процессов q , что у каждого соседа r процесса q какое-нибудь событие в r предшествует самому последнему событию, случившемуся в p .

Алгоритм Финна

Основная идея.

Множества Inc_p и $NInc_p$ формируются так.

1) Вначале $Inc_p = \{p\}$ и $NInc_p = \varnothing$.

Алгоритм Финна

Основная идея.

Множества Inc_p и $NInc_p$ формируются так.

1) Вначале $Inc_p = \{p\}$ и $NInc_p = \varnothing$.

2) Процесс p отправляет сообщения, содержащие Inc_p и $NInc_p$, всякий раз, когда одно из этих множеств расширяется.

Алгоритм Финна

Основная идея.

Множества Inc_p и $NInc_p$ формируются так.
1) Вначале $Inc_p = \{p\}$ и $NInc_p = \varnothing$.

2) Процесс p отправляет сообщения, содержащие Inc_p и $NInc_p$, всякий раз, когда одно из этих множеств расширяется.

3) Когда p получает сообщение с множествами Inc и $NInc$, полученные отличительные признаки добавляются к множествам Inc_p и $NInc_p$.

Алгоритм Финна

Основная идея.

Множества Inc_p и $NInc_p$ формируются так.

1) Вначале $Inc_p = \{p\}$ и $NInc_p = \varnothing$.

2) Процесс p отправляет сообщения, содержащие Inc_p и $NInc_p$, всякий раз, когда одно из этих множеств расширяется.

3) Когда p получает сообщение с множествами Inc и $NInc$, полученные отличительные признаки добавляются к множествам Inc_p и $NInc_p$.

4) Если p получил сообщения от всех своих соседей на входе, то отличительный признак p вставляется в $NInc_p$.

Алгоритм Финна

Основная идея.

Множества Inc_p и $NInc_p$ формируются так.

1) Вначале $Inc_p = \{p\}$ и $NInc_p = \varnothing$.

2) Процесс p отправляет сообщения, содержащие Inc_p и $NInc_p$, всякий раз, когда одно из этих множеств расширяется.

3) Когда p получает сообщение с множествами Inc и $NInc$, полученные отличительные признаки добавляются к множествам Inc_p и $NInc_p$.

4) Если p получил сообщения от всех своих соседей на входе, то отличительный признак p вставляется в $NInc_p$.

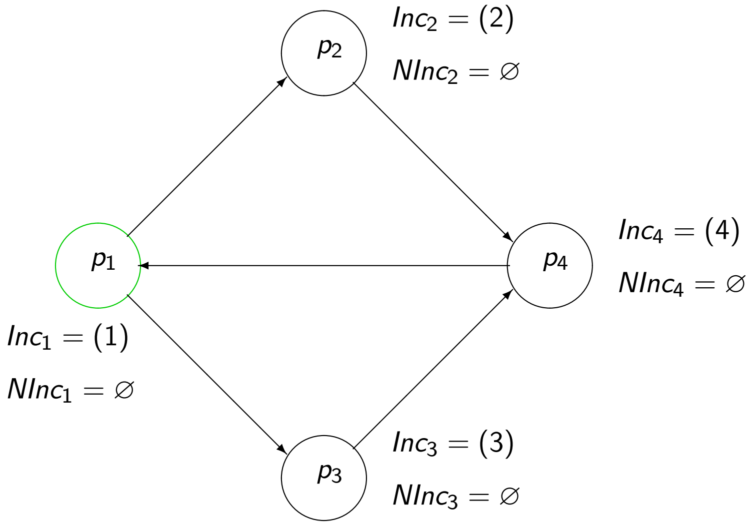
5) Если $Inc_p = NInc_p$, то p принимает решение.

Это означает, что каков бы ни был процесс q , если некоторое событие в q предшествует d_p , то у всякого соседа r процесса q также произошло какое-нибудь событие, предшествующее d_p . Отсюда следует, что для нашего алгоритма требование зависимости соблюдено.

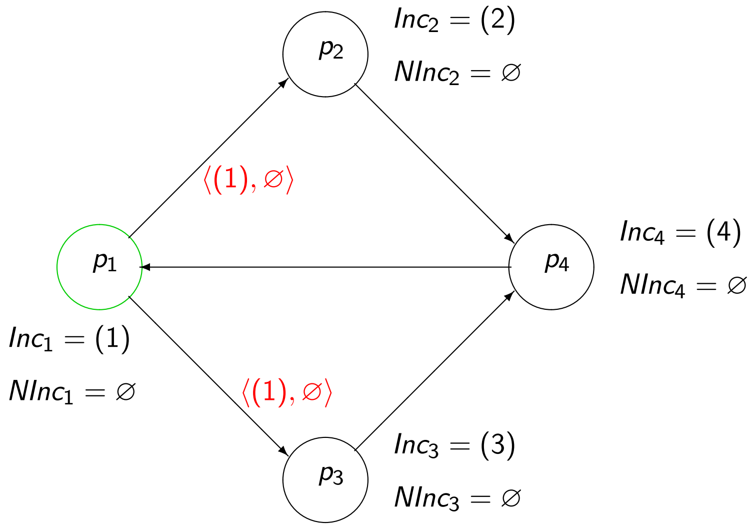
Алгоритм Финна

```
var Inc_p     : мн–во процессов           init {p} ;
    NInc_p    : мн–во процессов           init ∅ ;
    rec_p[q]  : bool for q ∈ In_p         init false ;
              (* индикаторы получения процессом p сообщения от q *)
begin if p is initiator then
          forall r ∈ Out_p do send ⟨sets, Inc_p, NInc_p⟩ to r ;
      while Inc_p ≠ NInc_p do
          begin receive ⟨sets, Inc, NInc⟩ from q_0 ;
                Inc_p := Inc_p ∪ Inc ; NInc_p := NInc_p ∪ NInc ;
                rec_p[q_0] := true ;
                if ∀q ∈ In_p : rec_p[q] then NInc_p := NInc_p ∪ {p} ;
                if Inc_p or NInc_p has changed then
                   forall r ∈ Out_p do send ⟨sets, Inc_p, NInc_p⟩ to r
          end;
      decide
end
```

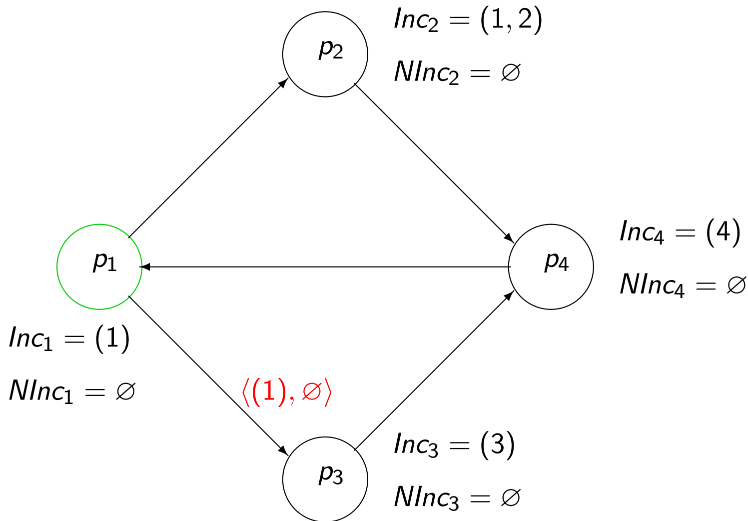
Фазовый алгоритм



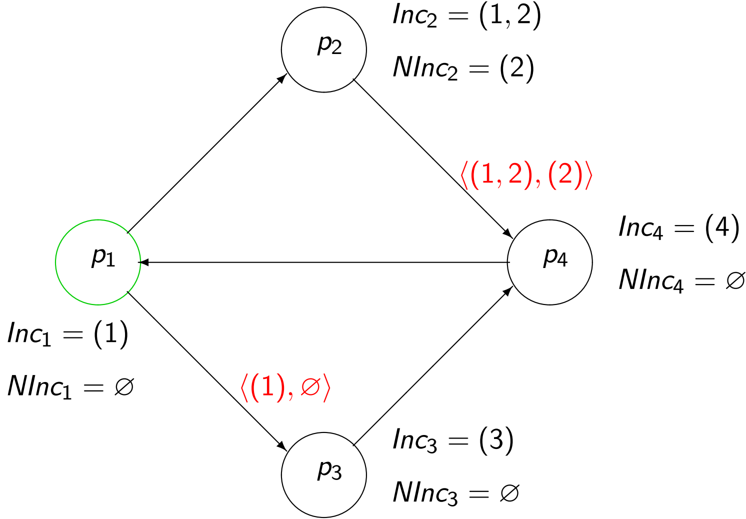
Фазовый алгоритм



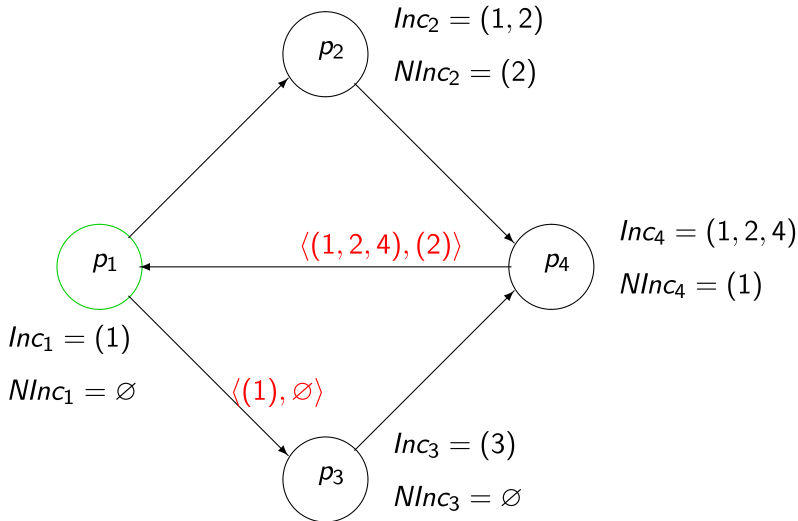
Фазовый алгоритм



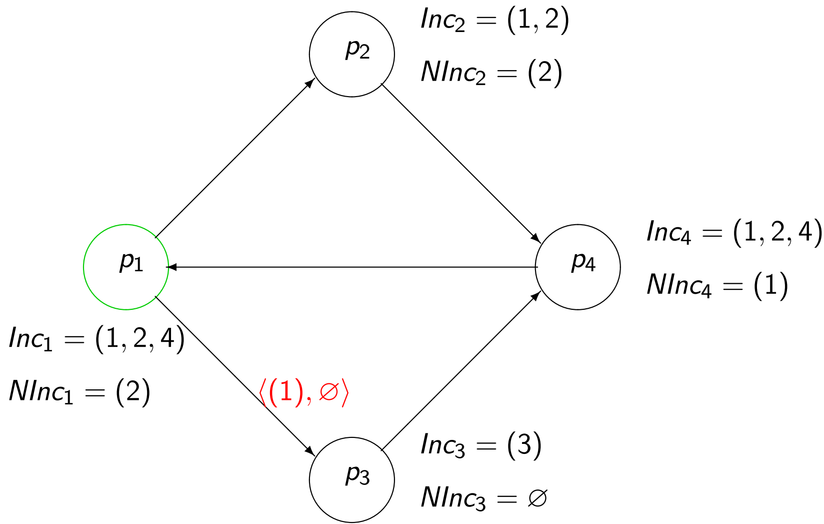
Фазовый алгоритм



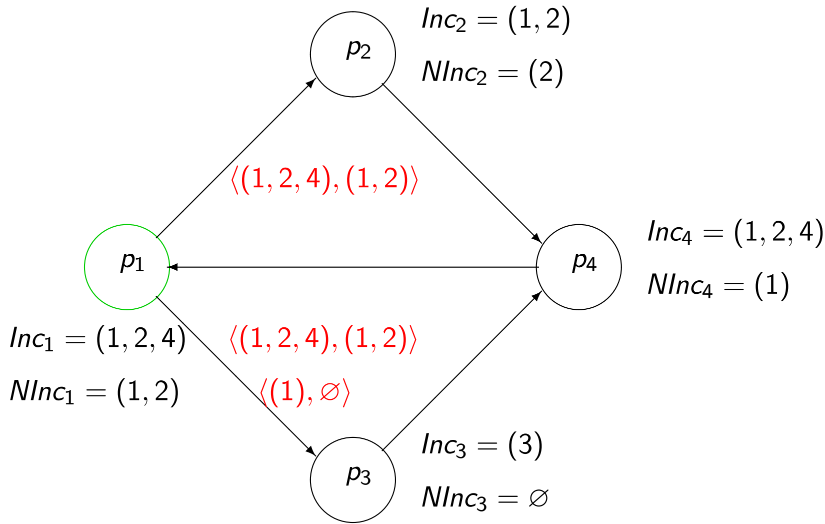
Фазовый алгоритм



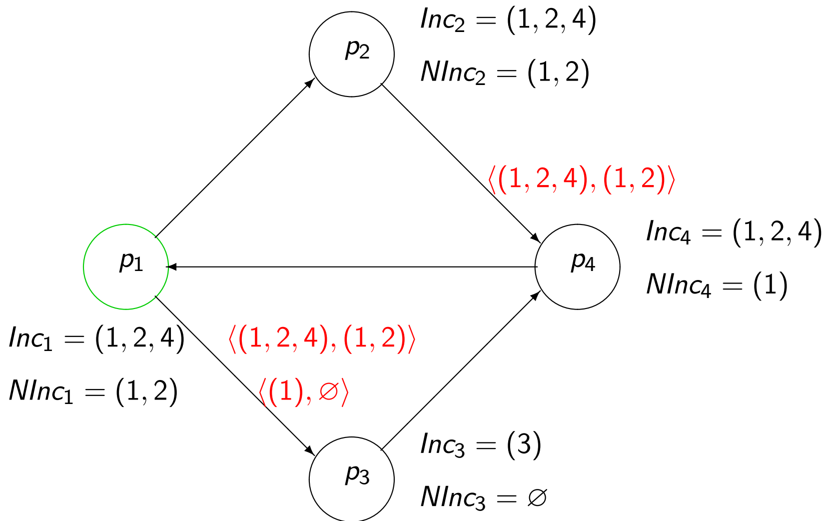
Фазовый алгоритм



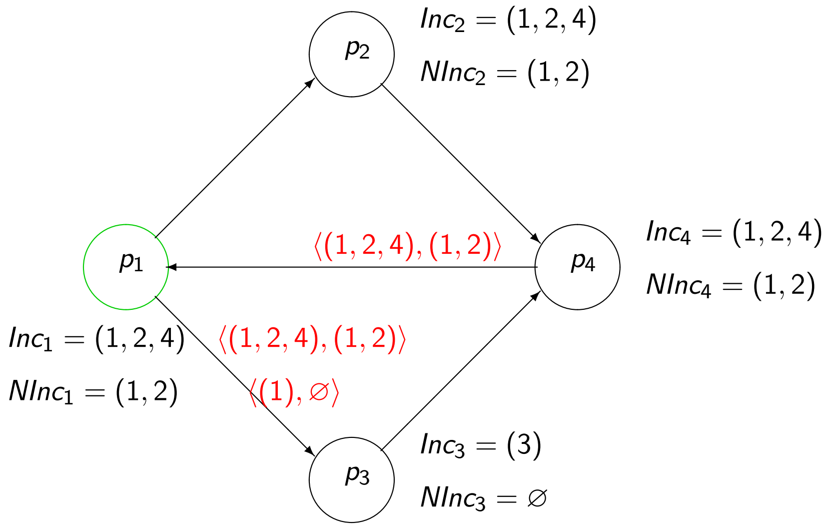
Фазовый алгоритм



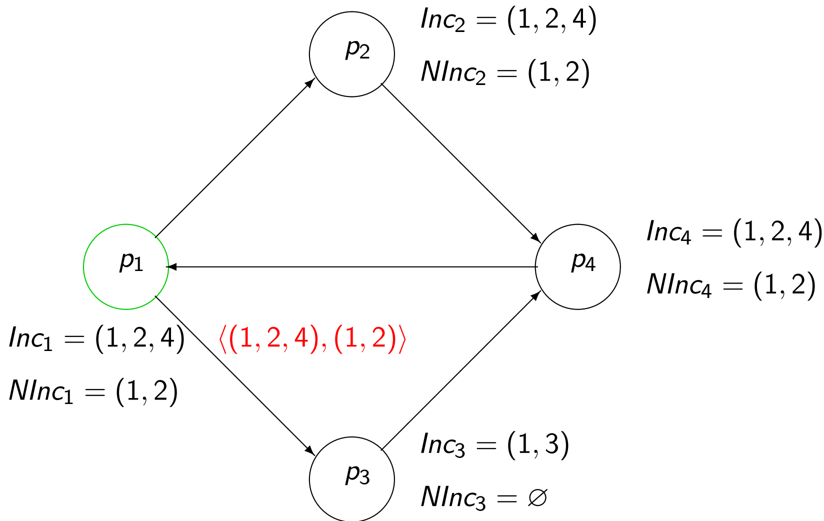
Фазовый алгоритм



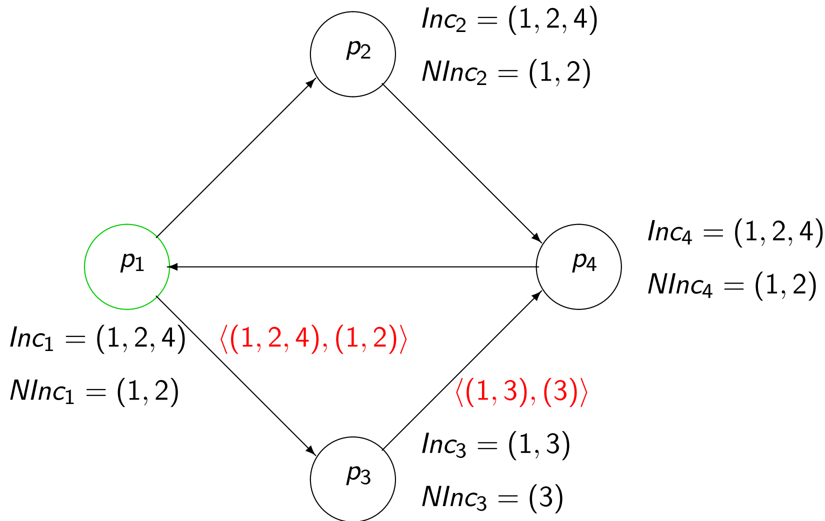
Фазовый алгоритм



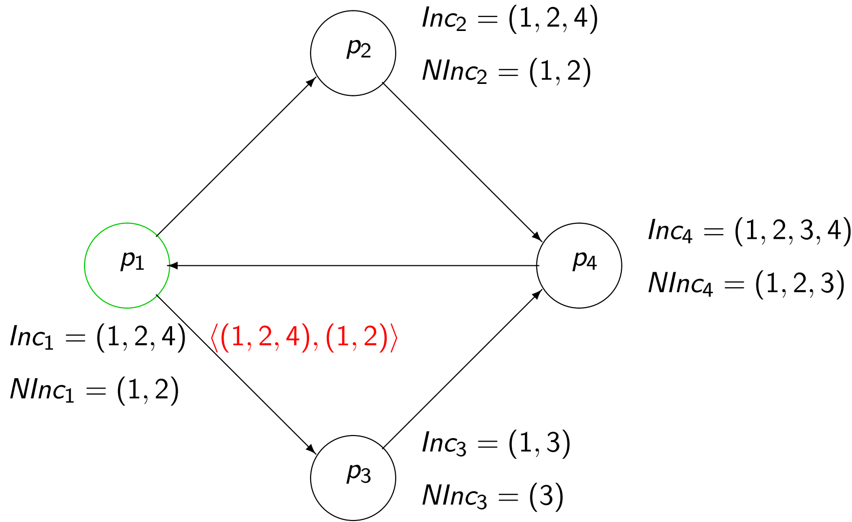
Фазовый алгоритм



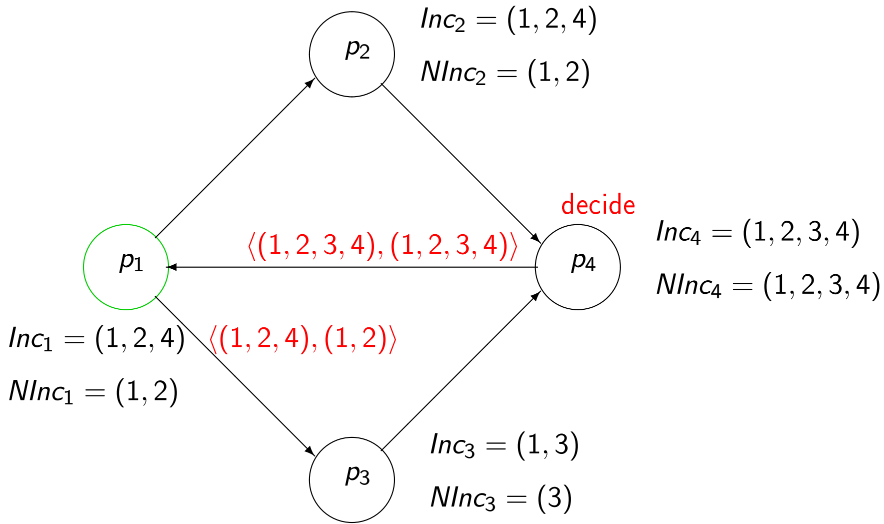
Фазовый алгоритм



Фазовый алгоритм



Фазовый алгоритм



Алгоритм Финна

Теорема об алгоритме Финна

Алгоритм Финна — это волновой алгоритм.

Доказательство.

Множества Inc_p и $NInc_p$ могут только увеличиваться. Их суммарный размер варьируется от 1 в первом сообщении до не более чем $2N$ в последнем сообщении. Значит, общее количество сообщений ограничено величиной $2N|E|$. Поэтому выполнения алгоритм завершаются.

Алгоритм Финна

Теорема об алгоритме Финна

Алгоритм Финна — это волновой алгоритм.

Доказательство.

Множества Inc_p и $NInc_p$ могут только увеличиваться. Их суммарный размер варьируется от 1 в первом сообщении до не более чем $2N$ в последнем сообщении. Значит, общее количество сообщений ограничено величиной $2N|E|$. Поэтому выполнения алгоритм завершаются.

Рассмотрим заключительную конфигурацию γ вычисления C . Как и в доказательстве предыдущей теоремы, можно показать, что если процесс p сумел отправить хотя бы одно сообщение (каждому соседу), и q — сосед процесса p на выходе, то q также сумел отправить хоть одно сообщение.

Отсюда следует, что каждый процесс сумел отправить хотя бы одно сообщение (по каждому каналу).

Алгоритм Финна

Доказательство.

Покажем, что в γ каждый процесс принял решение.

Алгоритм Финна

Доказательство.

Покажем, что в γ каждый процесс принял решение.

1) Для каждого ребра pq в конфигурации γ выполняется $Inc_p \subseteq Inc_q$, т.к., совершив последнее изменение множества Inc_p , процесс p отправил сообщение $\langle \mathbf{sets}, Inc_p, NInc_p \rangle$. Как только оно было получено, q выполнил оператор $Inc_q := Inc_q \cup Inc_p$.

Алгоритм Финна

Доказательство.

Покажем, что в γ каждый процесс принял решение.

1) Для каждого ребра pq в конфигурации γ выполняется $Inc_p \subseteq Inc_q$, т.к., совершив последнее изменение множества Inc_p , процесс p отправил сообщение $\langle \textbf{sets}, Inc_p, NInc_p \rangle$. Как только оно было получено, q выполнил оператор $Inc_q := Inc_q \cup Inc_p$.

Сильная связность сети приводит к тому, что $Inc_p = Inc_q$ для всех p и q . Т.к. $p \in Inc_p$, для каждого процесса p в конфигурации γ выполняется $Inc_p = \mathbb{P}$.

Алгоритм Финна

Доказательство.

2) Аналогичным образом можно показать, что для каждой пары процессов p и q выполняется $NInc_p = NInc_q$.

Алгоритм Финна

Доказательство.

2) Аналогичным образом можно показать, что для каждой пары процессов p и q выполняется $NInc_p = NInc_q$.

Поскольку каждый процесс успел отправить хотя бы одно сообщение по каждому каналу, для любого процесса p выполняется требование $\forall q \in In_p : rec_p[q]$. Следовательно, $p \in NInc_p$ для каждого p .

Алгоритм Финна

Доказательство.

2) Аналогичным образом можно показать, что для каждой пары процессов p и q выполняется $NInc_p = NInc_q$.

Поскольку каждый процесс успел отправить хотя бы одно сообщение по каждому каналу, для любого процесса p выполняется требование $\forall q \in In_p : rec_p[q]$. Следовательно, $p \in NInc_p$ для каждого p .

Отсюда следует, что $NInc_p = \mathbb{P}$ для каждого процесса p .

Из $Inc_p = \mathbb{P}$ и $NInc_p = \mathbb{P}$ вытекает, что $Inc_p = NInc_p$, и это означает, что в конфигурации γ каждый процесс p принял решение.

Алгоритм Финна

Доказательство.

Покажем, что решению d_p в процессе p предшествует какое-нибудь событие в каждом из процессов.

Алгоритм Финна

Доказательство.

Покажем, что решению d_p в процессе p предшествует какое-нибудь событие в каждом из процессов.

Для всякого события e в процессе p обозначим $Inc^{(e)}$ ($NInc^{(e)}$) значение Inc_p ($NInc_p$, соответственно) сразу после осуществления e . Следующие два утверждения обосновывают содержательный смысл множеств Inc_p и $NInc_p$.

Алгоритм Финна

Доказательство.

Покажем, что решению d_p в процессе p предшествует какое-нибудь событие в каждом из процессов.

Для всякого события e в процессе p обозначим $Inc^{(e)}$ ($NInc^{(e)}$) значение Inc_p ($NInc_p$, соответственно) сразу после осуществления e . Следующие два утверждения обосновывают содержательный смысл множеств Inc_p и $NInc_p$.

Утверждение 1.

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

Утверждение 2.

Если $q \in NInc^{(f)}$, то для всех $r \in In_q$ существует событие $e \in C_r : e \preceq f$.

Алгоритм Финна

Утверждение 1.

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

Обоснование.

Самостоятельно индукцией по длине причинно-следственной цепочки между событиями e и f.

Алгоритм Финна

Утверждение 2.

Если $q \in NInc^{(f)}$, то для всех $r \in In_q$ существует событие $e \in C_r : e \preceq f$.

Обоснование.

Пусть a_q обозначает внутреннее событие в процессе q , связанное с первым выполнением оператора $NInc_q := NInc_q \cup \{q\}$ процессом q . Событие a_q — это единственное событие, для которого выполняется $q \in NInc^{(a_q)}$, и которому не предшествует другое событие a' , удовлетворяющее условию $q \in NInc^{(a')}$. Значит, $q \in NInc^{(f)} \implies a_q \preceq f$.

Из описания алгоритма видно, что для каждого $r \in In_q$ найдется событие $e \in C_r$, предшествующее a_q . Отсюда следует доказываемое утверждение.

Алгоритм Финна

Доказательство.

Процесс p принимает решение только тогда, когда $Inc_p = NInc_p$. Пусть d_p — это событие решения в процессе p ; тогда $Inc^{(d_p)} = NInc^{(d_p)}$.

Алгоритм Финна

Доказательство.

Процесс p принимает решение только тогда, когда $Inc_p = NInc_p$. Пусть d_p — это событие решения в процессе p ; тогда $Inc^{(d_p)} = NInc^{(d_p)}$.

Теперь мы видим, что

1. $p \in Inc^{(d_p)}$;
2. из $q \in Inc^{(d_p)}$ следует $q \in NInc^{(d_p)}$, и отсюда получаем $In_q \subseteq Inc^{(d_p)}$.

Поскольку сеть сильно связная, мы получаем искомое равенство $Inc^{(d_p)} = \mathbb{P}$.

Алгоритмы обхода

Особый интерес представляют централизованные волновые алгоритмы, в которых все события линейно упорядочены по отношению причинно-следственной зависимости, и решение принимает инициатор. Волновые алгоритмы такого рода называются алгоритмами обхода .

Алгоритмы обхода

Особый интерес представляют централизованные волновые алгоритмы, в которых все события линейно упорядочены по отношению причинно-следственной зависимости, и решение принимает инициатор. Волновые алгоритмы такого рода называются алгоритмами обхода .

Алгоритмом обхода называется алгоритм, обладающий следующими тремя свойствами.

Алгоритмы обхода

Особый интерес представляют централизованные волновые алгоритмы, в которых все события линейно упорядочены по отношению причинно-следственной зависимости, и решение принимает инициатор. Волновые алгоритмы такого рода называются алгоритмами обхода .

Алгоритмом обхода называется алгоритм, обладающий следующими тремя свойствами.

1. В каждом вычислении единственный инициатор запускает алгоритм, отправляя одно-единственное сообщение.

Алгоритмы обхода

Особый интерес представляют централизованные волновые алгоритмы, в которых все события линейно упорядочены по отношению причинно-следственной зависимости, и решение принимает инициатор. Волновые алгоритмы такого рода называются алгоритмами обхода .

Алгоритмом обхода называется алгоритм, обладающий следующими тремя свойствами.

1. В каждом вычислении единственный инициатор запускает алгоритм, отправляя одно-единственное сообщение.
2. Всякий процесс после получения сообщения либо отправляет ровно одно сообщение, либо принимает решение.

Алгоритмы обхода

Особый интерес представляют централизованные волновые алгоритмы, в которых все события линейно упорядочены по отношению причинно-следственной зависимости, и решение принимает инициатор. Волновые алгоритмы такого рода называются алгоритмами обхода .

Алгоритмом обхода называется алгоритм, обладающий следующими тремя свойствами.

1. В каждом вычислении единственный инициатор запускает алгоритм, отправляя одно-единственное сообщение.
2. Всякий процесс после получения сообщения либо отправляет ровно одно сообщение, либо принимает решение.
3. Алгоритм завершает работу в инициаторе, и к тому моменту, когда это происходит, каждому процессу хотя бы один раз удалось отправить сообщение.

Алгоритм обхода произвольных сетей

Алгоритм обхода произвольных связных сетей был предложен Тарри в 1895. Этот алгоритм определяется двумя следующими правилами.

Алгоритм обхода произвольных сетей

Алгоритм обхода произвольных связных сетей был предложен Тарри в 1895. Этот алгоритм определяется двумя следующими правилами.

R1. Процесс не передает маркер по одному и тому же каналу дважды.

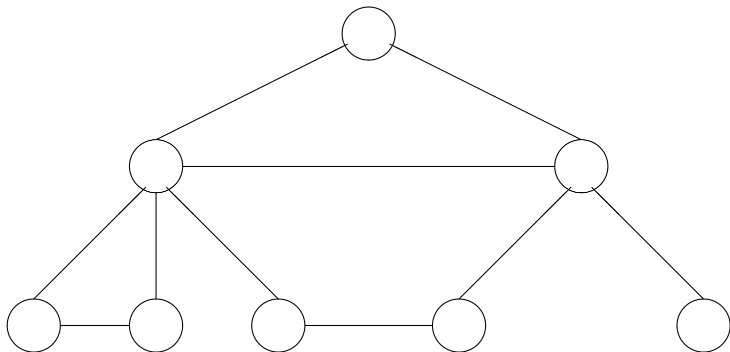
Алгоритм обхода произвольных сетей

Алгоритм обхода произвольных связных сетей был предложен Тарри в 1895. Этот алгоритм определяется двумя следующими правилами.

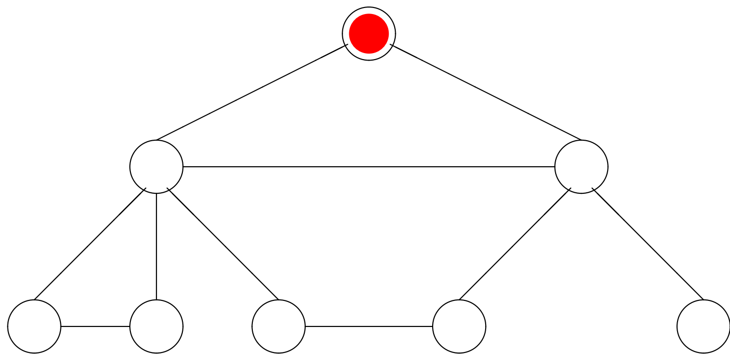
R1. Процесс не передает маркер по одному и тому же каналу дважды.

R2. Не-инициатор передает маркер своей родительской вершине (соседу, от которого он впервые получил маркер) только в том случае, когда его невозможно передать по другим каналам согласно правилу R1.

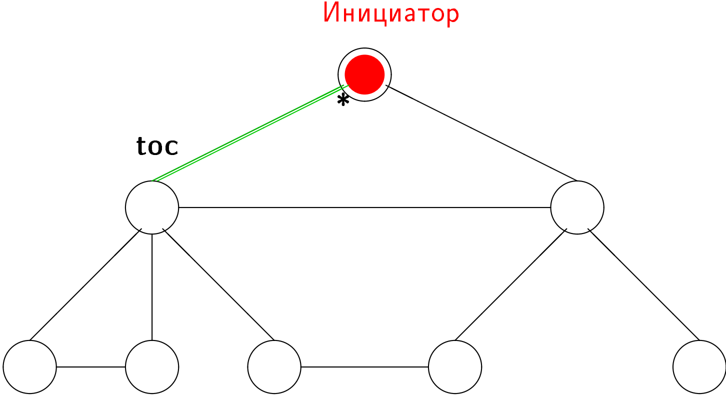
Алгоритм Тарри



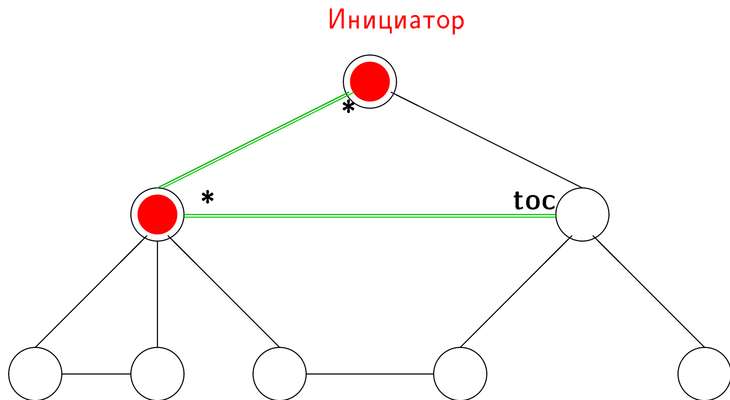
Алгоритм Тарри



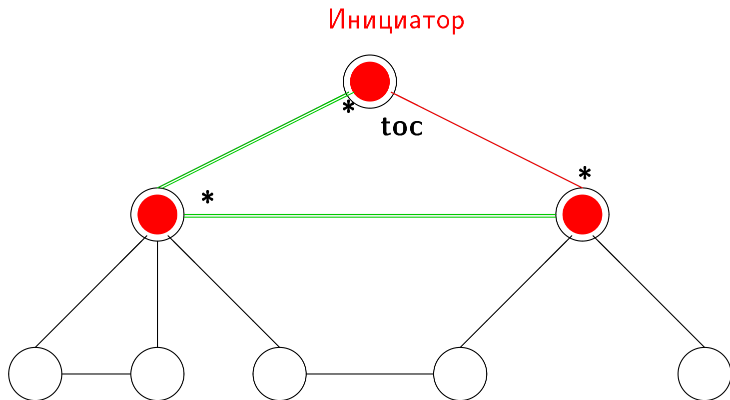
Алгоритм Тарри



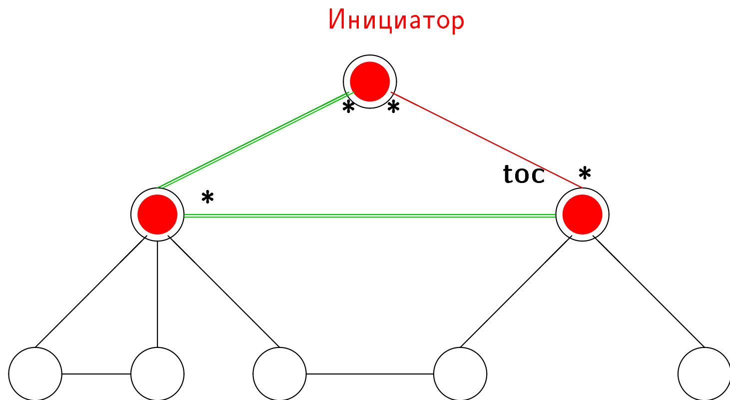
Алгоритм Тарри



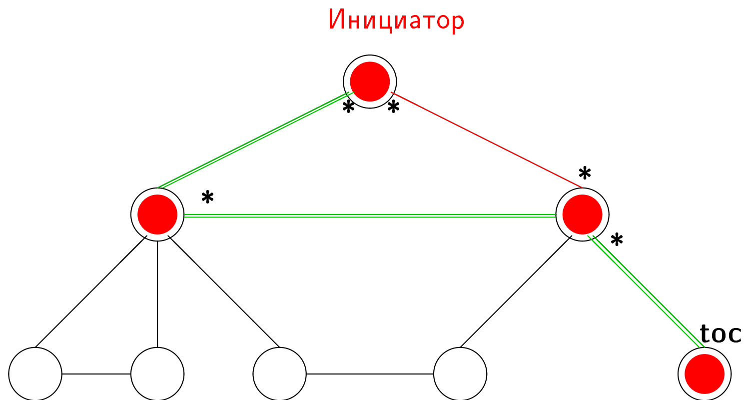
Алгоритм Тарри



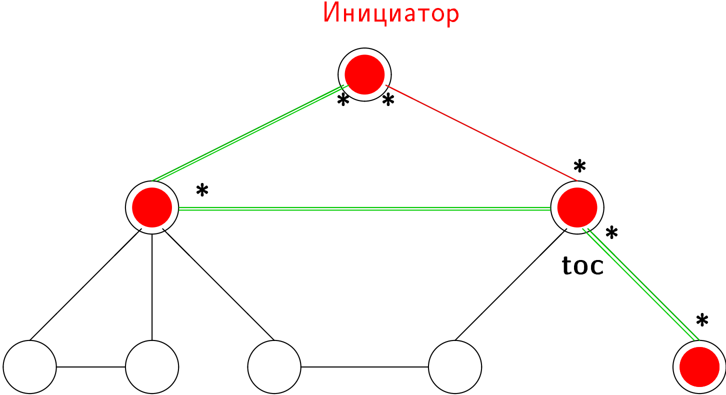
Алгоритм Тарри



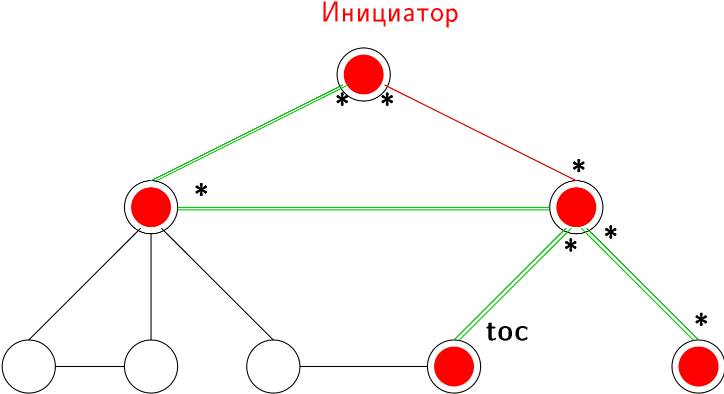
Алгоритм Тарри



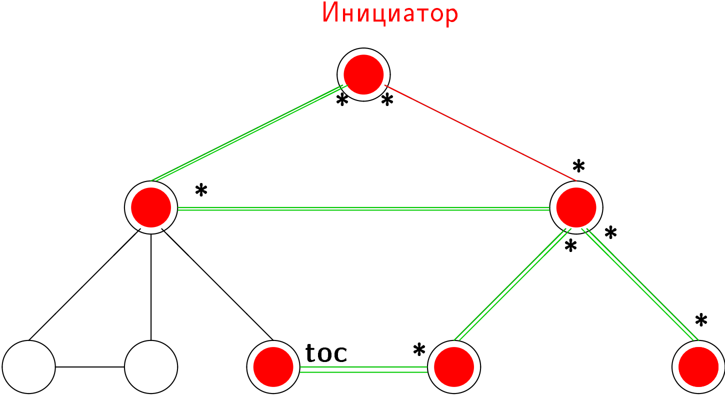
Алгоритм Тарри



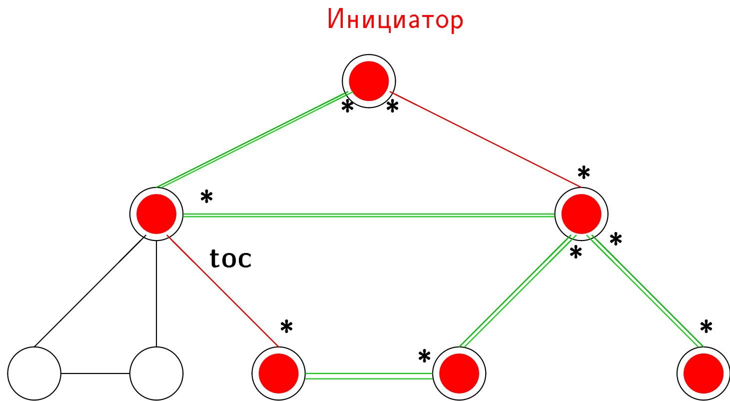
Алгоритм Тарри



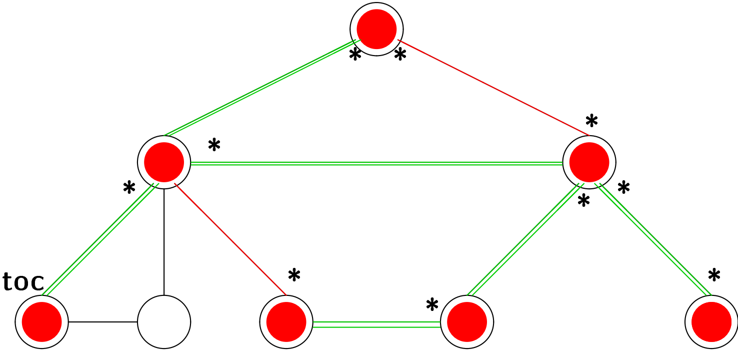
Алгоритм Тарри



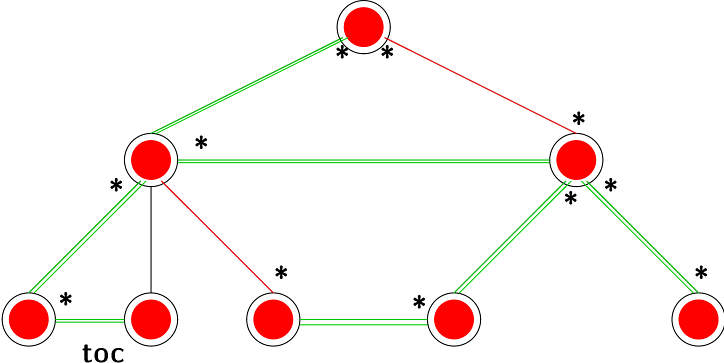
Алгоритм Тарри



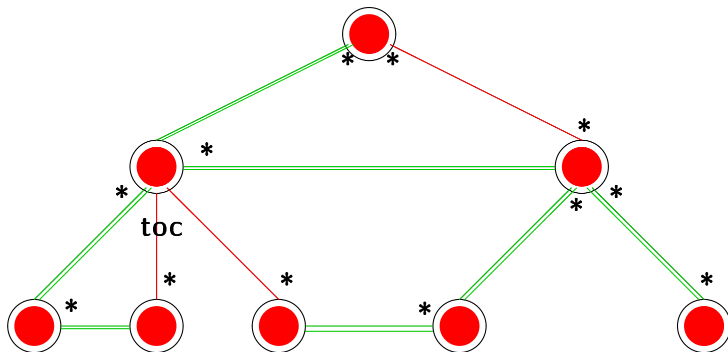
Алгоритм Тарри



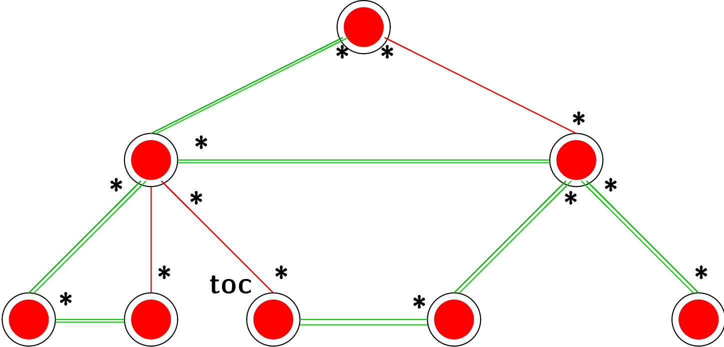
Алгоритм Тарри



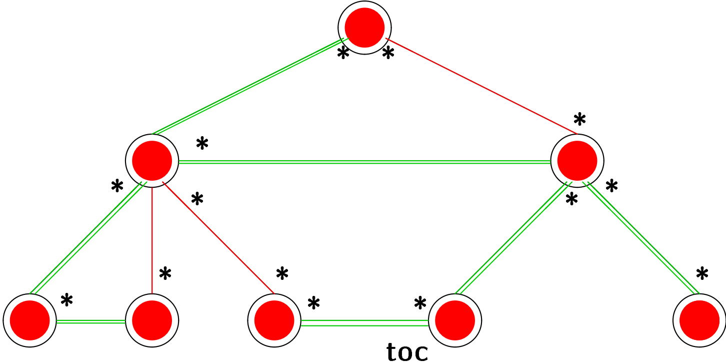
Алгоритм Тарри



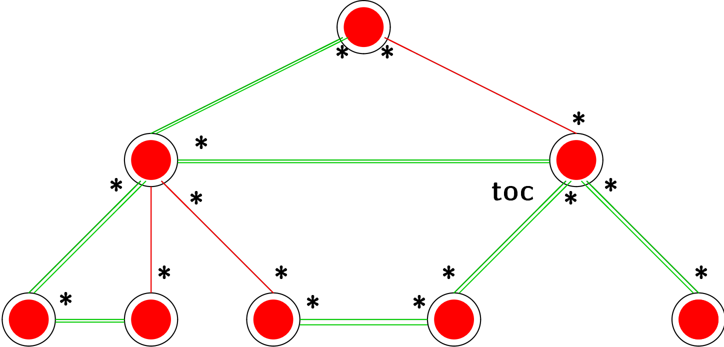
Алгоритм Тарри



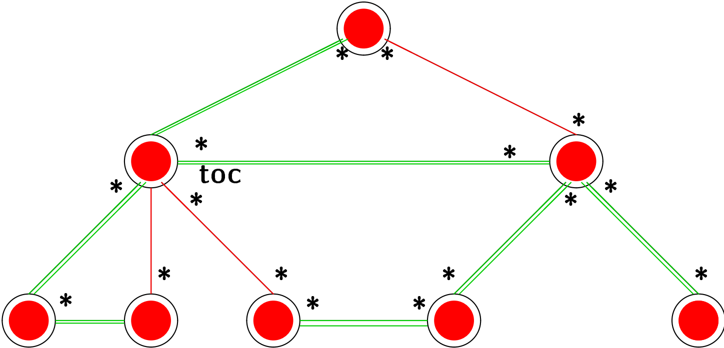
Алгоритм Тарри



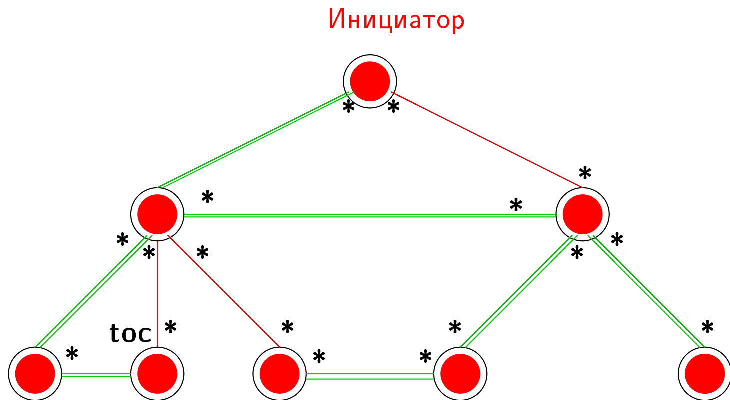
Алгоритм Тарри



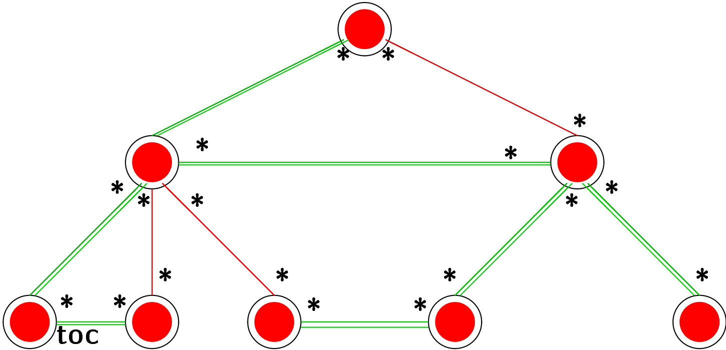
Алгоритм Тарри



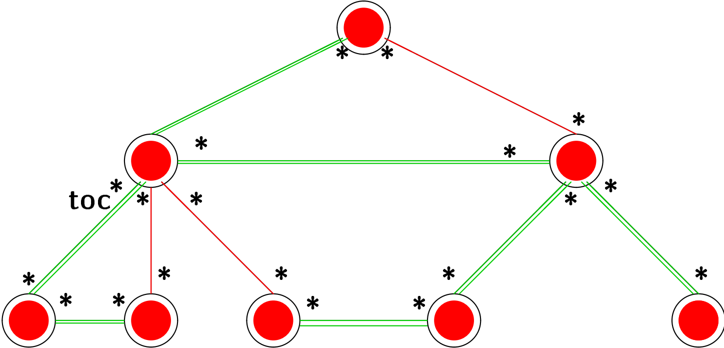
Алгоритм Тарри



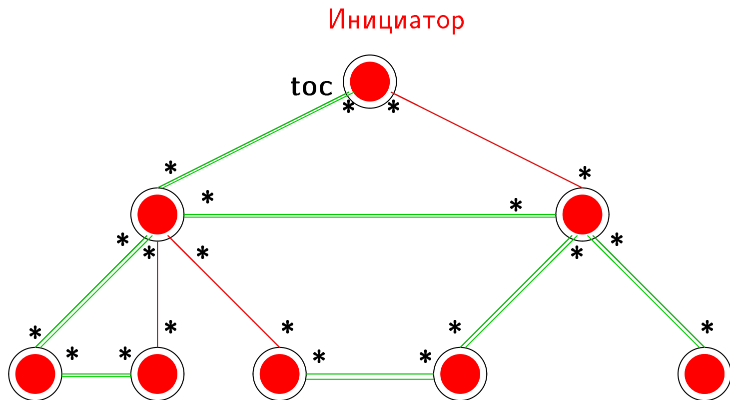
Алгоритм Тарри



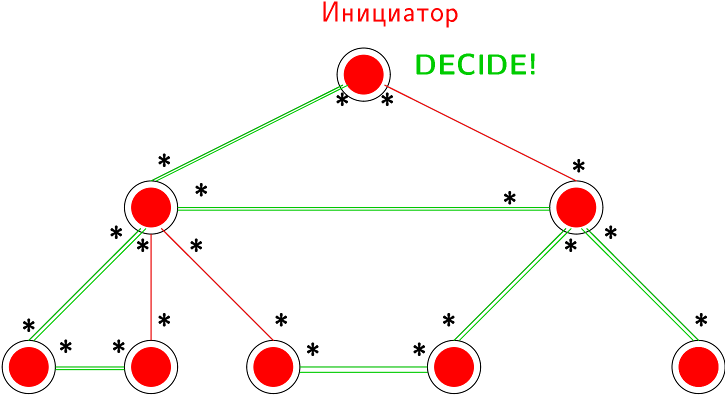
Алгоритм Тарри



Алгоритм Тарри



Алгоритм Тарри



Алгоритм Тарри

```
var used_p[q] : bool      init false for each q ∈ Neigh_p ;
    father_p  : process   init  udef ;
```

Только для инициатора, выполнить один раз:

begin $father_p := p$; choose $q \in Neigh_p$;
$used_p[q] := true$; send **tok** to q **end**

Для каждого процесса после приема **tok** от 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 choose $q \in Neigh_p \setminus \{father_p\}$ with $\neg used_p[q]$;
      $used_p[q] := true$; send **tok** to q
      end
   else begin $used_p[father_p] := true$; send **tok** to $father_p$
     end
end

Алгоритм Тарри

Теорема об алгоритме Тарри

Алгоритм Тарри — это алгоритм обхода.

Доказательство.

Так как маркер передается не более чем по одному разу в каждом из двух направлении по любому каналу, он совершает не более $2|E|$ переходов, пока алгоритм не завершится.

Алгоритм Тарри

Теорема об алгоритме Тарри

Алгоритм Тарри — это алгоритм обхода.

Доказательство.

Так как маркер передается не более чем по одному разу в каждом из двух направлении по любому каналу, он совершает не более $2|E|$ переходов, пока алгоритм не завершится.

Т.к. каждый процесс отправляет маркер по каждому каналу не более одного раза, то он его и получает по каждому каналу не более одного раза.

Алгоритм Тарри

Теорема об алгоритме Тарри

Алгоритм Тарри — это алгоритм обхода.

Доказательство.

Так как маркер передается не более чем по одному разу в каждом из двух направлении по любому каналу, он совершает не более $2|E|$ переходов, пока алгоритм не завершится.

Т.к. каждый процесс отправляет маркер по каждому каналу не более одного раза, то он его и получает по каждому каналу не более одного раза.

Если маркером владеет не-инициатор p , то процессу p пришлось получать маркер на один раз больше, чем отправлять его. Значит, число каналов, инцидентных p , превосходит число каналов, использованных этим процессом, по крайней мере на 1, и поэтому p не принимает решения, а передает маркер.

Алгоритм Тарри

Теорема об алгоритме Тарри

Алгоритм Тарри — это алгоритм обхода.

Доказательство.

Так как маркер передается не более чем по одному разу в каждом из двух направлении по любому каналу, он совершает не более $2|E|$ переходов, пока алгоритм не завершится.

Т.к. каждый процесс отправляет маркер по каждому каналу не более одного раза, то он его и получает по каждому каналу не более одного раза.

Если маркером владеет не-инициатор p , то процессу p пришлось получать маркер на один раз больше, чем отправлять его. Значит, число каналов, инцидентных p , превосходит число каналов, использованных этим процессом, по крайней мере на 1, и поэтому p не принимает решения, а передает маркер.

Значит решение может принять только инициатор.

Алгоритм Тарри

Доказательство.

Все каналы, инцидентные инициатору были использованы по одному разу в каждом направлении.

Алгоритм Тарри

Доказательство.

Все каналы, инцидентные инициатору были использованы по одному разу в каждом направлении.

Каждый канал был использован инициатором для передачи маркера, ибо в противном случае алгоритм не завершается.

Инициатору пришлось получать маркер столько же раз, сколько и отправлять его.

А поскольку получать его приходилось по разным каналам, то маркер был получен по одному разу с использованием каждого канала.

Алгоритм Тарри

Доказательство.

Для каждого посещенного маркером процесса p все каналы, инцидентные p , использовались для передачи маркера в каждом направлении по одному разу.

Алгоритм Тарри

Доказательство.

Для каждого посещенного маркером процесса p все каналы, инцидентные p , использовались для передачи маркера в каждом направлении по одному разу.

Допустим противное. Пусть p — самый первый из посещенных маркером процессов, для которого не выполняется это предположение. Таким процессом не может быть инициатор.

Алгоритм Тарри

Доказательство.

Для каждого посещенного маркером процесса p все каналы, инцидентные p , использовались для передачи маркера в каждом направлении по одному разу.

Допустим противное. Пусть p — самый первый из посещенных маркером процессов, для которого не выполняется это предположение. Таким процессом не может быть инициатор.

Согласно выбору p все каналы, инцидентные $father_p$, были использованы по одному разу в каждом направлении. Поэтому p уже отправил маркер в родительскую вершину.

Алгоритм Тарри

Доказательство.

Для каждого посещенного маркером процесса p все каналы, инцидентные p , использовались для передачи маркера в каждом направлении по одному разу.

Допустим противное. Пусть p — самый первый из посещенных маркером процессов, для которого не выполняется это предположение. Таким процессом не может быть инициатор.

Согласно выбору p все каналы, инцидентные $father_p$, были использованы по одному разу в каждом направлении. Поэтому p уже отправил маркер в родительскую вершину.

Значит, p использовал все свои каналы для отправления маркера. Т.к. маркер вернулся к инициатору, процессу p пришлось получать его столь же часто, сколь и отправлять.

Алгоритм Тарри

Доказательство.

Для каждого посещенного маркером процесса p все каналы, инцидентные p , использовались для передачи маркера в каждом направлении по одному разу.

Допустим противное. Пусть p — самый первый из посещенных маркером процессов, для которого не выполняется это предположение. Таким процессом не может быть инициатор.

Согласно выбору p все каналы, инцидентные $father_p$, были использованы по одному разу в каждом направлении. Поэтому p уже отправил маркер в родительскую вершину.

Значит, p использовал все свои каналы для отправления маркера. Т.к. маркер вернулся к инициатору, процессу p пришлось получать его столь же часто, сколь и отправлять.

Но тогда по каждому инцидентному каналу процесс p получал маркер по одному разу. А это противоречит сделанному предположению.

Алгоритм Тарри

Доказательство.

Маркер посетил все процессы.

Алгоритм Тарри

Доказательство.

Маркер посетил все процессы.

Если бы нашлись процессы, которые маркеру не удалось посетить, то это означало бы, что для некоторой пары соседних процессов p и q маркеру удалось побывать в p , но не удалось посетить q .

Но это противоречит тому, что процессу p пришлось использовать каждый канал в обоих направлениях. Значит маркер побывал во всех процессах, и все каналы были задействованы по одному разу в каждом направлении.

Алгоритм Тарри

Каждое вычисление алгоритма Тарри задает в сети остовное дерево (почему?).

Корнем дерева служит инициатор, и каждый не-инициатор p к моменту завершения вычисления уже заносит в переменную $father_p$ ссылку на родительскую вершину в этом дереве.

Если требуется, чтобы каждый процесс обладал (к моменту завершения вычисления) сведениями о том, какие из его соседей являются сыновними вершинами в этом дереве, то этого можно достичь отправляя специальное сообщение процессу $father_p$.

Алгоритмы обхода в глубину

Особо интересны алгоритмы построения остовных деревьев, у которых каждое стягивающее ребро соединяет две вершины, одна из которых выступает в роли предшественника другой.

Стягивающее ребро — это ребро, не принадлежащее остовному дереву.

Для заданного остовного дерева T сети G и для каждой его вершины p мы запись $T[p]$ будет обозначать множества вершин поддерева, растущего из p , а запись $A[p]$ — множество предшественников p , лежащих в дереве T на пути из корня в p . Заметим, что $q \in T[p] \iff p \in A[q]$.

Алгоритмы обхода в глубину

Особо интересны алгоритмы построения остовных деревьев, у которых каждое стягивающее ребро соединяет две вершины, одна из которых выступает в роли предшественника другой.

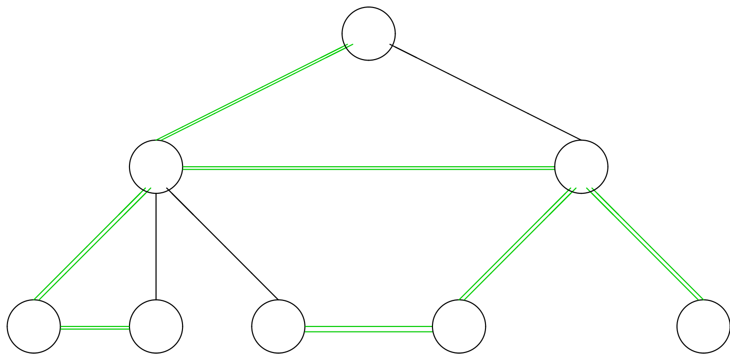
Стягивающее ребро — это ребро, не принадлежащее остовному дереву.

Для заданного остовного дерева T сети G и для каждой его вершины p мы запись $T[p]$ будет обозначать множества вершин поддерева, растущего из p , а запись $A[p]$ — множество предшественников p , лежащих в дереве T на пути из корня в p . Заметим, что $q \in T[p] \iff p \in A[q]$.

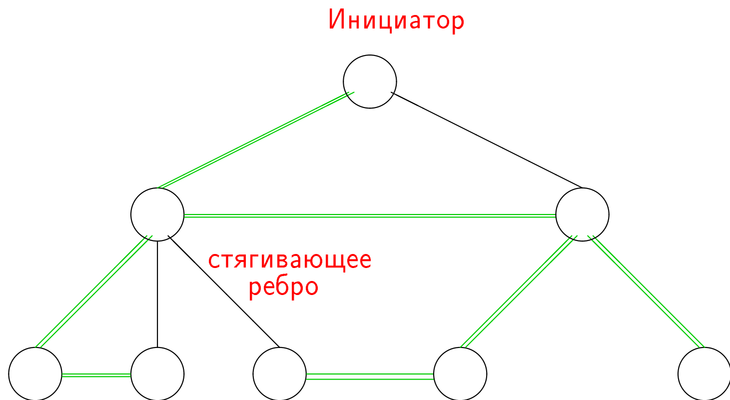
Определение

Корневое остовное дерево T сети G называется деревом поиска в глубину , если для каждого стягивающего ребра pq выполняется соотношение $q \in T[p] \vee q \in A[p]$.

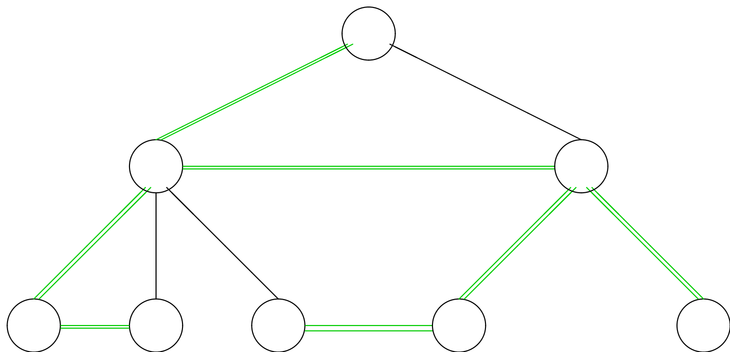
Алгоритмы обхода в глубину



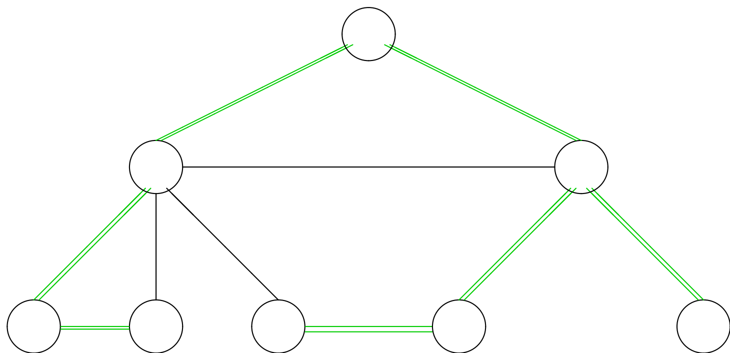
Алгоритмы обхода в глубину



Алгоритмы обхода в глубину

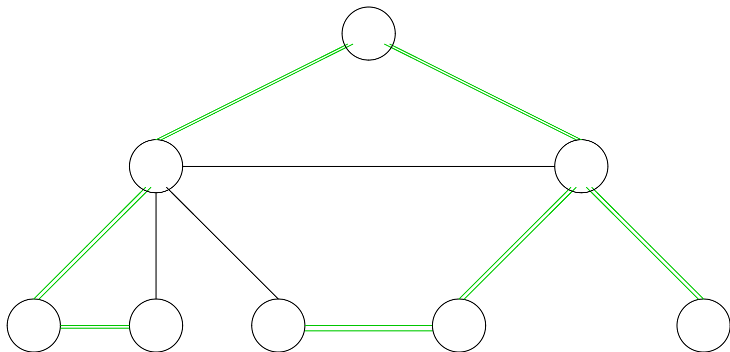


Алгоритмы обхода в глубину



Алгоритмы обхода в глубину

Не является деревом обхода в глубину



Алгоритмы обхода в глубину

Задача.

Приведите пример сети, для которой остовное дерево, построенное алгоритмом Тарри, не является деревом поиска в глубину.

Классический алгоритм обхода в глубину

Классический алгоритм поиска в глубину получается из алгоритма Тарри, в котором свобода выбора очередного соседа для передачи ему маркера ограничена третьим правилом следующего вида.

R3. Всякий раз, когда процесс получает маркер, он возвращает его назад по тому же каналу, если это не противоречит правилам R1 и R2.

Классический алгоритм обхода в глубину

var $used_p[q]$: **bool** **init** false for each $q \in Neigh_p$;
$father_p$: process **init** udef ;

Только для инициатора, выполнить один раз:

begin $father_p := p$; choose $q \in Neigh_p$;
$used_p[q] := true$; send **tok** to q **end**

Для каждого процесса после приема **tok** от 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 **tok** to q
end
else begin $used_p[father_p] := true$; send **tok** to $father_p$ **end**

end

Классический алгоритм обхода в глубину

Теорема о классическом алгоритме обхода в глубину

Классический алгоритм поиска в глубину строит остовное дерево поиска в глубину, используя при этом $2|E|$ обменов сообщениями.

Доказательство.

Поскольку в основу нашего алгоритма положен алгоритм Тарри, мы имеем дело с алгоритмом обхода, который вычисляет остовное дерево T . Как было уже установлено, по каждому каналу передаются два сообщения (по одному в каждом направлении), и это служит обоснование сложности по числу сообщений.

Классический алгоритм обхода в глубину

Теорема о классическом алгоритме обхода в глубину

Классический алгоритм поиска в глубину строит остовное дерево поиска в глубину, используя при этом $2|E|$ обменов сообщениями.

Доказательство.

Поскольку в основу нашего алгоритма положен алгоритм Тарри, мы имеем дело с алгоритмом обхода, который вычисляет остовное дерево T . Как было уже установлено, по каждому каналу передаются два сообщения (по одному в каждом направлении), и это служит обоснование сложности по числу сообщений.

Остается убедиться в том, что дерево, построенное в результате применения правила R3, будет деревом поиска в глубину.

Классический алгоритм обхода в глубину

Доказательство.

Согласно правилу R3 вслед за первым прохождением по стягивающиему ребру немедленно следует повторное прохождение в обратном направлении.

Классический алгоритм обхода в глубину

Доказательство.

Согласно правилу R3 вслед за первым прохождением по стягивающему ребру немедленно следует повторное прохождение в обратном направлении.

Допустим, что pq — это стягивающее ребро, и p — первый процесс, воспользовавшийся этим ребром. К тому моменту, когда процесс q получает маркер от p , этот маркер уже побывал в q (иначе q занес бы в переменную $father_q$ ссылку на p , и наше ребро не могло бы считаться стягивающим) и $used_q[p]$ имеет значение **false** (поскольку по предположению p был первым из этих двух процессов, воспользовавшимся указанным ребром).

Следовательно, согласно правилу R3, q немедленно возвращает маркер процессу p .

Классический алгоритм обхода в глубину

Доказательство.

Теперь можно показать, что **если ребро pq является стягивающим, и его впервые использовал процесс p , то** $q \in A[p]$.

Классический алгоритм обхода в глубину

Доказательство.

Теперь можно показать, что **если ребро pq является стягивающим, и его впервые использовал процесс p , то** $q \in A[p]$.

Рассмотрим путь, по которому следовал маркер до тех пор, пока он не был отправлен по ребру pq . Т.к. pq — стягивающее ребро, маркер уже побывал в вершине q до того, как он достиг q по указанному ребру: $\ldots, q, \ldots, p, q$

Сократим этот путь, заменяя все фрагменты вида r_1, r_2, r_1 , где $r_1 r_2$ — стягивающее ребро, фрагментом r_1 . Тогда все стягивающие ребра будут удалены. Полученный в результате путь будет путем в T , состоящим только из ребер, использованных прежде первого обращения к pq .

Если q не является предшественником p , то ребро из q в $father_q$ было задействовано ранее, нежели было использовано ребро qp , вопреки правилу R2 нашего алгоритма. □

Алгоритм Авербаха

Классический алгоритм поиска в глубину имеет большую сложность по времени, т.к. все ребра сети обходятся последовательно. При этом маркер-сообщение обходит все стягивающие ребра и немедленно возвращается. Во всяком решение более низкой сложности по времени маркер-сообщение проходит только по ребрам, входящим в дерево.

Алгоритм Авербаха

Классический алгоритм поиска в глубину имеет большую сложность по времени, т.к. все ребра сети обходятся последовательно. При этом маркер-сообщение обходит все стягивающие ребра и немедленно возвращается. Во всяком решение более низкой сложности по времени маркер-сообщение проходит только по ребрам, входящим в дерево.

В алгоритме Авербаха процесс при передаче маркера уже знает о том, у какого из его соседей уже побывал маркер.

Тогда процесс либо выбирает какого-нибудь соседа, не получавшего до сих пор маркер, либо возвращает маркер в родительскую вершину, если такого соседа нет.

Алгоритм Авербаха

Когда маркер впервые достается процессу p (для инициатора — в момент запуска алгоритма), p оповещает каждого соседа r, кроме родительской вершины, отправляя ему сообщение $\langle \mathbf{vis} \rangle$.

Маркер не передается, до тех пор пока p не получит сообщение $\langle \mathbf{ack} \rangle$ от всех своих соседей. Это гарантирует, что каждый сосед r процесса p осведомлен к моменту отправления маркера процессом p о том, что этот маркер уже побывал у p.

Когда позднее маркер достигнет r, процесс r уже не будет отправлять маркер процессу p, если только p не является родительской вершиной для r.

Алгоритм Авербаха

var $used_p[q]$: **bool** **init** false for each $q \in Neigh_p$;
(*Указывает, отправлял ли p маркер процессу q*)
$father_p$: process **init** udef ;

Только для инициатора, выполнить один раз:

begin $father_p := p$; choose $q \in Neigh_p$;
forall $r \in Neigh_p$ **do** send $\langle$**vis**$\rangle$ to r ;
forall $r \in Neigh_p$ **do** receive $\langle$**ack**$\rangle$ from r ;
$used_p[q] := true$; send $\langle$**tok**$\rangle$ to q
end

Для каждого процесса после получения ⟨**tok**⟩ от 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
```

Для каждого процесса после получения ⟨**vis**⟩ от q_0:

```
begin used_p[q_0] := true ; send ⟨ack⟩ to q_0 end
```

Алгоритм Авербаха

Теорема о алгоритме Авербаха

Алгоритм Авербаха строит дерево поиска в глубину за $4N-2$ единиц времени, проводя при этом $4|E|$ обменов сообщениями.

Алгоритм Сидона

В алгоритме Сидона не отправляются сообщения **ack**, и за счет этого улучшается сложность по времени. Маркер отправляется немедленно, без задержки на две единицы времени для ожидания подтверждений.

Здесь возможно возникновение следующей ситуации. В процессе p уже побывал маркер, и этот процесс уже отправил сообщение **vis** своему соседу r .

Позднее маркер достается процессу r , но еще до того, как **vis** от p достигло r . Тогда r может передать маркер процессу p по стягивающему ребру.

Алгоритм Сидона

Чтобы справиться с такой ситуацией, процесс p записывает (в переменную $last_p$) тех соседей, которым он в последнюю очередь отправлял маркер.

Когда маркер проходит только по ребрам дерева, процесс p получает в очередной раз маркер от того самого соседа $last_p$. А в том сценарии, который был описан выше, p получает маркер от другого соседа, а именно от r . **В этом случае маркер игнорируется**, но p помечает ребро rp как использованное, как будто по нему было получено сообщение **vis** от r .

Процесс r получает от p сообщение **vis**, после того как он передал маркер процессу p , т. е. r получает сообщение **vis** от своего соседа $last_r$. Тогда r ведет себя так, как если бы он вообще не передавал маркер процессу p , а именно, r выбирает следующего соседа и передает ему маркер.

Алгоритм Сидона

var $used_p[q]$: **bool** **init** false for each $q \in Neigh_p$;
(* Указывает, отправлял ли p маркер процессу q *)
$father_p$: process **init** udef ;
$last_p$: process **init** udef ;

Только для инициатора, выполнить один раз:

begin $father_p := p$; choose $q \in Neigh_p$;
forall $r \in Neigh_p$ **do** send **vis** to r ;
$used_p[q] := true$; $last_p := q$; send **tok** to q
end

Для каждого процесса после получения **vis** от q_0:

begin $used_p[q_0] := true$;
if $q_0 = last_p$ **then** (* Истолковывать как сообщение **tok**
forward **tok** message as upon receipt of **tok** message
end

Для каждого процесса после получения **tok** от q_0:

```
begin if last_p ≠ udef and last_p ≠ q_0 then used_p[q_0] := true
      (* Это стягивающее ребро, истолковывать как сообщение vis *)
      else(* Поступать, как в предыдущем алгоритме *)
          begin if father_p = udef then
                      begin father_p := q_0 ;
                              forall r ∈ Neigh_p \ {father_p} do send vis to r
                      end ;
                if p — инициатор 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 ; last_p := q; send tok to q
                        end
                   else begin used_p[father_p] := true; send tok to father_p
                        end
      end
```

Алгоритм Сидона

Теорема о алгоритме Сидона

Алгоритм Сидона строит DFS дерево за $2N-2$ единицы времени, используя $4|E|$ обменов сообщениями.

Задачи

1. Предположим, что есть желание использовать волновой алгоритм в сети, в которой возможно **дублирование** сообщений.
   - Какие изменения нужно внести в алгоритм эха?
   - Какие изменения нужно внести в алгоритм Финна?
2. Адаптируйте алгоритм эха для вычисления суммы входных данных всех процессов.

КОНЕЦ ЛЕКЦИИ 7.