Распределенные алгоритмы

ЛЕКТОР: В.А. Захаров

Лекция 6.

Волновые алгоритмы: определения, основные свойства, область применения.
Древесный алгоритм.
Алгоритм эха.

Зачем нужны волновые алгоритмы

При проектировании распределенных алгоритмов для разных приложений некоторые общие проблемы, которые нужно уметь решать как промежуточные задачи. К ним относятся

- широковещательное распространение информация (например, сообщения о начале или завершении вычисления),
- установление глобальной синхронизации между процессами,
- запуск выполнения некоторого действия в каждом процессе,
- вычисление функции при условии, что каждый из процессов содержит только часть входных данных,
- и многие другие.

Эти задачи можно решать путем обмена сообщениями согласно некоторой предписанной схеме, которая зависит от топологии сети и позволяет задействовать все процессы.

Соглашение о сетевой модели

Далее в этой главе мы будем полагать, что сеть

- имеет фиксированную топологию (никаких изменений топологии не происходит);

Соглашение о сетевой модели

Далее в этой главе мы будем полагать, что сеть

- имеет фиксированную топологию (никаких изменений топологии не происходит);
- является неориентированной (по каждому каналу сообщения могут передаваться в обоих направлениях);

Соглашение о сетевой модели

Далее в этой главе мы будем полагать, что сеть

- имеет фиксированную топологию (никаких изменений топологии не происходит);
- является неориентированной (по каждому каналу сообщения могут передаваться в обоих направлениях);
- является связной (между любыми двумя процессами пролегает путь),

Соглашение о сетевой модели

Далее в этой главе мы будем полагать, что сеть

- имеет фиксированную топологию (никаких изменений топологии не происходит);
- является неориентированной (по каждому каналу сообщения могут передаваться в обоих направлениях);
- является связной (между любыми двумя процессами пролегает путь),
- является асинхронной,

Соглашение о сетевой модели

Далее в этой главе мы будем полагать, что сеть

- имеет фиксированную топологию (никаких изменений топологии не происходит);
- является неориентированной (по каждому каналу сообщения могут передаваться в обоих направлениях);
- является связной (между любыми двумя процессами пролегает путь),
- является асинхронной,
- не имеет доступа к показаниям вещественных часов глобального времени.

Обозначения

- $\mathbb{P}$ — множество всех процессов системы;

Обозначения

- $\mathbb{P}$ — множество всех процессов системы;
- E — множество всех каналов системы;

Обозначения

- $\mathbb{P}$ — множество всех процессов системы;
- E — множество всех каналов системы;
- C — вычисление системы;

Обозначения

- $\mathbb{P}$ — множество всех процессов системы;
- E — множество всех каналов системы;
- C — вычисление системы;
- $\preceq$ — отношение причинно-следственного предшествования;

Обозначения

- $\mathbb{P}$ — множество всех процессов системы;
- E — множество всех каналов системы;
- C — вычисление системы;
- $\preceq$ — отношение причинно-следственного предшествования;
- C_p — множество событий процесса p в вычислении C ;

Обозначения

- $\mathbb{P}$ — множество всех процессов системы;
- E — множество всех каналов системы;
- C — вычисление системы;
- $\preceq$ — отношение причинно-следственного предшествования;
- C_p — множество событий процесса p в вычислении C ;
- $|C|$ — количество событий в вычислении C .

Обозначения

- $\mathbb{P}$ — множество всех процессов системы;
- E — множество всех каналов системы;
- C — вычисление системы;
- $\preceq$ — отношение причинно-следственного предшествования;
- C_p — множество событий процесса p в вычислении C ;
- $|C|$ — количество событий в вычислении C .

Предполагается, что существуют внутренние события специального типа, которые называются событиями решения ; такие события просто представлены оператором *decide* .

Обозначения

- $\mathbb{P}$ — множество всех процессов системы;
- E — множество всех каналов системы;
- C — вычисление системы;
- $\preceq$ — отношение причинно-следственного предшествования;
- C_p — множество событий процесса p в вычислении C ;
- $|C|$ — количество событий в вычислении C .

Предполагается, что существуют внутренние события специального типа, которые называются событиями решения ; такие события просто представлены оператором *decide* .

В волновом алгоритме проводится обмен конечным числом сообщений, а затем принимается решение, которое должно иметь причинно-следственную зависимость хотя бы от одного события в каждом из процессов системы.

Определение волнового алгоритма

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

Определение волнового алгоритма

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

1. Завершение. Каждое вычисление конечно:

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

Определение волнового алгоритма

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:

$\forall C : \exists e \in C : e$ является событием решения.

Определение волнового алгоритма

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:

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

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

$$\forall C : \forall e \in C : (e \text{ — событие решения } \Rightarrow \forall q \in C : \exists f \in C_q : f \preceq e).$$

Определение волнового алгоритма

Вычисление волнового алгоритма называется волной .

При выполнении волнового алгоритма проводится различие процессами:

1. инициаторы (или иначе, стартовые процессы) запускают выполнение своего локального алгоритма самопроизвольно, т. е. алгоритм запускается по некоторому условию внутри процесса.
2. неинициаторы (или иначе, последователи) вовлекаются в распределенный алгоритм, только когда по ходу вычисления поступает сообщение, которое запускает выполнение алгоритма процесса.

Таким образом, первое событие инициатора — это внутреннее событие или отправление сообщения, а первое событие неинициатора — это событие приема сообщения.

Определение волнового алгоритма

Параметры волновых алгоритмов.

1. Централизация . Алгоритм называется централизованным , если у каждого вычисления есть в точности один инициатор, и децентрализованным , если алгоритм может быть запущен спонтанно некоторым произвольным подмножеством процессов.

Определение волнового алгоритма

Параметры волновых алгоритмов.

1. Централизация . Алгоритм называется централизованным , если у каждого вычисления есть в точности один инициатор, и децентрализованным , если алгоритм может быть запущен спонтанно некоторым произвольным подмножеством процессов.
2. Топология . Алгоритм может быть спроектирован в расчете на специальную топологию, наподобие кольца, дерева, клики и т.п.

Определение волнового алгоритма

Параметры волновых алгоритмов.

1. Централизация . Алгоритм называется централизованным , если у каждого вычисления есть в точности один инициатор, и децентрализованным , если алгоритм может быть запущен спонтанно некоторым произвольным подмножеством процессов.
2. Топология . Алгоритм может быть спроектирован в расчете на специальную топологию, наподобие кольца, дерева, клики и т.п.
3. Первоначальные сведения . Каждый процесс может знать свое собственное уникальное имя или имена своих соседей;

Определение волнового алгоритма

Параметры волновых алгоритмов.

1. Централизация . Алгоритм называется централизованным , если у каждого вычисления есть в точности один инициатор, и децентрализованным , если алгоритм может быть запущен спонтанно некоторым произвольным подмножеством процессов.
2. Топология . Алгоритм может быть спроектирован в расчете на специальную топологию, наподобие кольца, дерева, клики и т.п.
3. Первоначальные сведения . Каждый процесс может знать свое собственное уникальное имя или имена своих соседей;
4. Число решений . Решение принимает только один процесс или все процессы.

Определение волнового алгоритма

Параметры волновых алгоритмов.

1. Централизация . Алгоритм называется централизованным , если у каждого вычисления есть в точности один инициатор, и децентрализованным , если алгоритм может быть запущен спонтанно некоторым произвольным подмножеством процессов.
2. Топология . Алгоритм может быть спроектирован в расчете на специальную топологию, наподобие кольца, дерева, клики и т.п.
3. Первоначальные сведения . Каждый процесс может знать свое собственное уникальное имя или имена своих соседей;
4. Число решений . Решение принимает только один процесс или все процессы.
5. Сложность . Мерами сложности служат количество обменов сообщениями, число битов информации, отправленных при обмене сообщениями, и время, затрачиваемое одним вычислением.

Свойства волновых алгоритмов

Утверждение 1.

Для каждого события $e \in C$ существуют такой инициатор p и такое событие $f \in C_p$, что $f \preceq e$.

Свойства волновых алгоритмов

Утверждение 1.

Для каждого события $e \in C$ существуют такой инициатор p и такое событие $f \in C_p$, что $f \preceq e$.

Доказательство.

Выберем в качестве f минимальный элемент в истории события e . Такой элемент f существует, поскольку история каждого события конечна. Покажем, что процесс p, в котором происходит событие f , является инициатором.

Свойства волновых алгоритмов

Утверждение 1.

Для каждого события $e \in C$ существуют такой инициатор p и такое событие $f \in C_p$, что $f \preceq e$.

Доказательство.

Выберем в качестве f минимальный элемент в истории события e . Такой элемент f существует, поскольку история каждого события конечна. Покажем, что процесс p, в котором происходит событие f , является инициатором.
Затметим, что f — это первое событие процесса p. Первое событие неинициатора — это событие приема, которому должно предшествовать соответствующее событие отправления, что противоречит минимальности f . Значит, p — инициатор. $\square$

Свойства волновых алгоритмов

Пусть C — централизованная волна с инициатором p. Для каждого неинициатора q обозначим символом $father_q$ того соседа процесса q, от которого q получил первое сообщение.

Свойства волновых алгоритмов

Пусть C — централизованная волна с инициатором p. Для каждого неинициатора q обозначим символом $father_q$ того соседа процесса q, от которого q получил первое сообщение.

Утверждение 2.

Граф $T = (\mathbb{P}, E_T)$, где $E_T = \{\langle q, r\rangle : q \neq p \wedge r = father_q\}$, является остовным деревом, направленным к p.

Свойства волновых алгоритмов

Пусть C — централизованная волна с инициатором p. Для каждого неинициатора q обозначим символом $father_q$ того соседа процесса q, от которого q получил первое сообщение.

Утверждение 2.

Граф $T = (\mathbb{P}, E_T)$, где $E_T = \{\langle q, r\rangle : q \neq p \wedge r = father_q\}$, является остовным деревом, направленным к p.

Доказательство.

Число вершин в T превосходит число дуг на единицу.
Поэтому достаточно показать, что T не содержит циклов.
И это действительно так, поскольку для e_q, первого события в q, наличие дуги $qr \in E_T$ влечет $e_r \preceq e_q$, а отношение $\preceq$ является частичным порядком. $\square$

Свойства волновых алгоритмов

Утверждение 3.

Пусть C — волна и $decide \in C_p$. Тогда
$\forall q \neq p : \exists f \in C_q : (f \preceq decide \wedge f$ — отправление сообщения$)$.

Свойства волновых алгоритмов

Утверждение 3.

Пусть C — волна и $decide \in C_p$. Тогда
$\forall q \neq p : \exists f \in C_q : (f \preceq decide \wedge f$ — отправление сообщения$)$.

Доказательство.

Так как C — волна, существуют такие события $f \in C_q$, что $f \preceq decide$. Выберем в качестве f **последнее** событие в C_q , предшествующее $decide$.
Покажем, что f — событие отправления сообщения.

Свойства волновых алгоритмов

Утверждение 3.

Пусть C — волна и $decide \in C_p$. Тогда
$\forall q \neq p : \exists f \in C_q : (f \preceq decide \wedge f$ — отправление сообщения$)$.

Доказательство.

Так как C — волна, существуют такие события $f \in C_q$, что $f \preceq decide$. Выберем в качестве f **последнее** событие в C_q , предшествующее $decide$.
Покажем, что f — событие отправления сообщения.
Из определения $\preceq$ вытекает существование такой причинно-следственной цепочки $f = e_0, e_1, \ldots, e_k = decide$, что для каждого $i < k$ события e_i и e_{i+1} являются либо последовательными событиями в одном процессе, либо парой соответствующих событий отправления-приема сообщения.

Свойства волновых алгоритмов

Утверждение 3.

Пусть C — волна и $decide \in C_p$. Тогда
$\forall q \neq p : \exists f \in C_q : (f \preceq decide \wedge f$ — отправление сообщения$)$.

Доказательство.

Так как C — волна, существуют такие события $f \in C_q$, что $f \preceq decide$. Выберем в качестве f **последнее** событие в C_q , предшествующее $decide$.
Покажем, что f — событие отправления сообщения.
Из определения $\preceq$ вытекает существование такой причинно-следственной цепочки $f = e_0, e_1, \ldots, e_k = decide$, что для каждого $i < k$ события e_i и e_{i+1} являются либо последовательными событиями в одном процессе, либо парой соответствующих событий отправления-приема сообщения.
Т.к. f — последнее событие в процессе q , предшествующее событию $decide$, событие e_1 происходит в процессе, отличном от q . Значит, f — событие отправления сообщения. $\square$

Сложность волновых алгоритмов

Утверждение 4.

Пусть C — такая волна с одним инициатором p в распределенной системе, состоящей из N процессов, что в p происходит событие решения *decide* . Тогда в вычислении C происходит обмен по крайней мере N сообщениями.

Сложность волновых алгоритмов

Утверждение 4.

Пусть C — такая волна с одним инициатором p в распределенной системе, состоящей из N процессов, что в p происходит событие решения *decide* . Тогда в вычислении C происходит обмен по крайней мере N сообщениями.

Доказательство.

По утверждению 1 каждому событию в C предшествует некоторое событие в процессе p. Поэтому в p произойдет хотя бы одно событие отправления сообщения. По утверждению 2 событие отправления сообщения также произойдет и в каждом другом процессе, а это означает, что произойдет как минимум N обменов сообщениями. $\square$

Сложность волновых алгоритмов

Утверждение 5.

Пусть A — волновой алгоритм для произвольной сети, в котором не используются первоначальные сведения об отличительных признаках соседей. Тогда A в каждом вычислении проводит не менее $|E|$ обменов сообщениями.

Сложность волновых алгоритмов

Утверждение 5.

Пусть A — волновой алгоритм для произвольной сети, в котором не используются первоначальные сведения об отличительных признаках соседей. Тогда A в каждом вычислении проводит не менее $|E|$ обменов сообщениями.

Доказательство.

Предположим, что A имеет вычисление C, в котором проводится менее $|E|$ обменов сообщениями, т. е. по некоторому каналу xy не передаются сообщения.

Сложность волновых алгоритмов

Утверждение 5.

Пусть A — волновой алгоритм для произвольной сети, в котором не используются первоначальные сведения об отличительных признаках соседей. Тогда A в каждом вычислении проводит не менее $|E|$ обменов сообщениями.

Доказательство.

Предположим, что A имеет вычисление C, в котором проводится менее $|E|$ обменов сообщениями, т. е. по некоторому каналу xy не передаются сообщения. Рассмотрим сеть G', полученную за счет добавления одной точки z в канале между x и y.

Сложность волновых алгоритмов

Утверждение 5.

Пусть A — волновой алгоритм для произвольной сети, в котором не используются первоначальные сведения об отличительных признаках соседей. Тогда A в каждом вычислении проводит не менее $|E|$ обменов сообщениями.

Доказательство.

Предположим, что A имеет вычисление C, в котором проводится менее $|E|$ обменов сообщениями, т. е. по некоторому каналу xy не передаются сообщения.
Рассмотрим сеть G', полученную за счет добавления одной точки z в канале между x и y.
Поскольку точки не идентифицируют своих соседей, начальное состояние x и y в G' будет тем же самым, что и в сети G. Это справедливо и для всех остальных точек G.

Сложность волновых алгоритмов

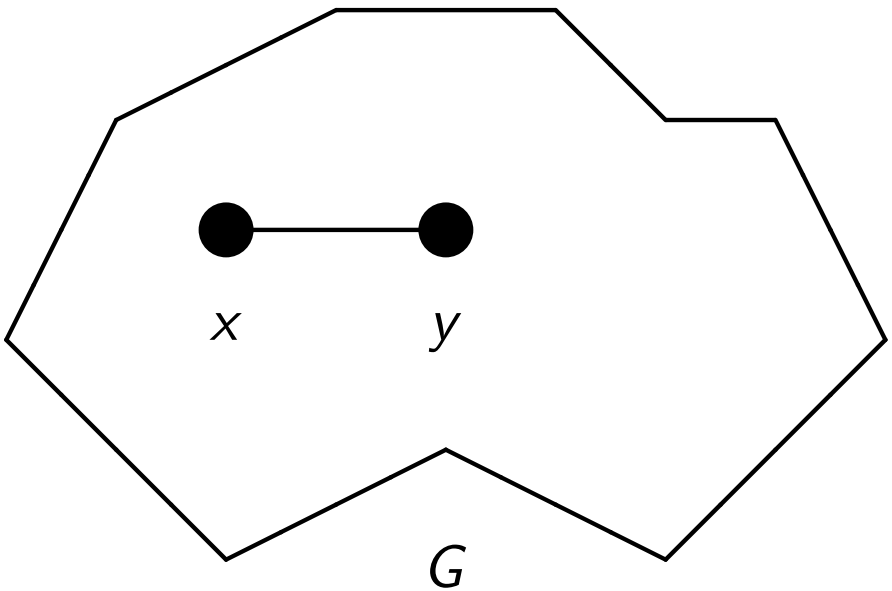
Утверждение 5.

Пусть A — волновой алгоритм для произвольной сети, в котором не используются первоначальные сведения об отличительных признаках соседей. Тогда A в каждом вычислении проводит не менее $|E|$ обменов сообщениями.

Доказательство.

Предположим, что A имеет вычисление C, в котором проводится менее $|E|$ обменов сообщениями, т. е. по некоторому каналу xy не передаются сообщения.
Рассмотрим сеть G', полученную за счет добавления одной точки z в канале между x и y.
Поскольку точки не идентифицируют своих соседей, начальное состояние x и y в G' будет тем же самым, что и в сети G. Это справедливо и для всех остальных точек G.
Следовательно, все события в C произойдут в том же самом порядке начиная с исходной конфигурации G', но теперь событию *decide* не предшествует никакое событие в z. □

Задача маршрутизации



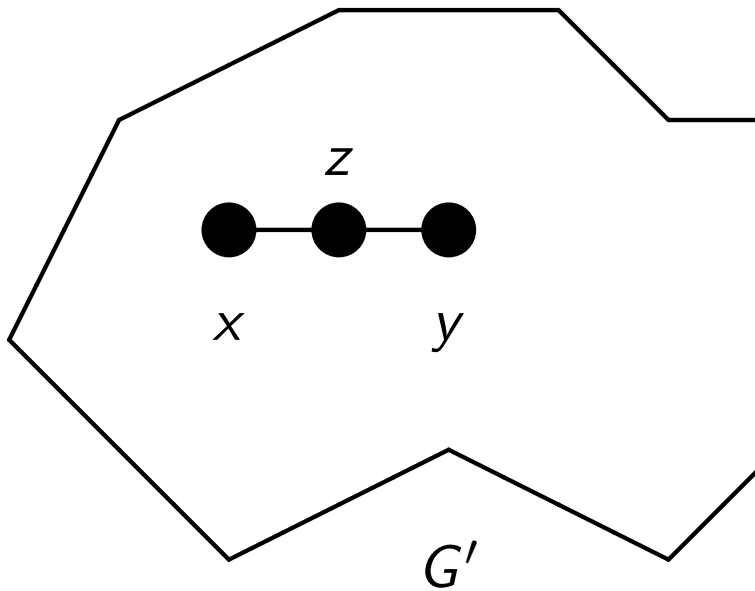


Рис.: Вставка процесса в неиспользуемый канал

Применение волновых алгоритмов: PIF

Волновые алгоритмы необходимы для широковещательного распространения информации с подтверждением о завершении выполнения (широковещательной связи Propagation of Information with Feedback, PIF).

Пусть имеется подмножество процессов, обладающих некоторым сообщением M . Нужно, чтобы все процессы приняли M . Некоторые выделенные процессы должны быть уведомлены о завершении широковещательной связи; это означает, что в них должно быть выработано специальное событие notify с тем условием, что событие notify произойдет только в том случае, когда все процессы уже получат сообщение M .

Применение волновых алгоритмов: PIF

Утверждение 6.

Каждый PIF-алгоритм — это волновой алгоритм.

Применение волновых алгоритмов: PIF

Утверждение 6.

Каждый PIF-алгоритм — это волновой алгоритм.

Доказательство.

Пусть P — это некоторый PIF-алгоритм. Тогда каждое вычисление алгоритма P конечно и в каждом вычислении происходит уведомляющее событие (*notify*).
Если в некотором вычислении алгоритма P происходит уведомление d_P , которому не предшествует никакое событие в процессе q , то по теореме о перестановке независимых событий существует такое выполнение алгоритма P , в котором уведомление происходит, до того как q получит какое-либо сообщение, что противоречит нашим требованиям. $\square$

Применение волновых алгоритмов: PIF

Утверждение 6.

Каждый PIF-алгоритм — это волновой алгоритм.

Доказательство.

Пусть P — это некоторый PIF-алгоритм. Тогда каждое вычисление алгоритма P конечно и в каждом вычислении происходит уведомляющее событие (*notify*).
Если в некотором вычислении алгоритма P происходит уведомление d_P , которому не предшествует никакое событие в процессе q , то по теореме о перестановке независимых событий существует такое выполнение алгоритма P , в котором уведомление происходит, до того как q получит какое-либо сообщение, что противоречит нашим требованиям. □

Это утверждение справедливо только для систем с асинхронной передачей сообщений (Почему?).

Применение волновых алгоритмов: PIF

Утверждение 7.

Каждый волновой алгоритм может быть использован в качестве PIF-алгоритма.

Применение волновых алгоритмов: PIF

Утверждение 7.

Каждый волновой алгоритм может быть использован в качестве PIF-алгоритма.

Доказательство.

Пусть A — волновой алгоритм. Чтобы представить A как PIF-алгоритм, процессы, первоначально хранящие M, следует считать инициаторами. Информацию M нужно добавить к каждому сообщению, передаваемому в алгоритме A.
Не-инициаторы пассивны, до тех пор пока сами они не получат хоть одно сообщение, а значит так же станут осведомлены о M.
Событию *decide* в волне предшествует по крайней мере одно событие в каждом процессе. Следовательно, когда событие решения произойдет, каждый процесс будет уже осведомлен о M, и это можно расценивать как требуемое PIF-алгоритмом уведомление *notify* . □

Применение волновых алгоритмов: SYN

Волновые алгоритмы необходимы для глобальной синхронизации (Synchronization, SYN) между процессами.
Требование синхронизации таковы: в каждом процессе q должно быть выполнено событие a_q , и в некоторых процессах событие b_p должно быть выполнено так, чтобы выполнение a_q произошло по времени ранее, нежели выполнение любого из событий b_p .
В SYN-алгоритме события b_p рассматриваются как события *decide* .

Применение волновых алгоритмов: SYN

Волновые алгоритмы необходимы для глобальной синхронизации (Synchronization, SYN) между процессами.
Требование синхронизации таковы: в каждом процессе q должно быть выполнено событие a_q , и в некоторых процессах событие b_p должно быть выполнено так, чтобы выполнение a_q произошло по времени ранее, нежели выполнение любого из событий b_p .
В SYN-алгоритме события b_p рассматриваются как события *decide* .

Утверждение 8.

Каждый волновой алгоритм может быть использован в качестве SYN-алгоритма.

Применение волновых алгоритмов: SYN

Волновые алгоритмы необходимы для глобальной синхронизации (Synchronization, SYN) между процессами.
Требование синхронизации таковы: в каждом процессе q должно быть выполнено событие a_q , и в некоторых процессах событие b_p должно быть выполнено так, чтобы выполнение a_q произошло по времени ранее, нежели выполнение любого из событий b_p .
В SYN-алгоритме события b_p рассматриваются как события *decide* .

Утверждение 8.

Каждый волновой алгоритм может быть использован в качестве SYN-алгоритма.

Утверждение 9.

Каждый SYN-алгоритм может быть использован в качестве волнового алгоритма.

Применение волновых алгоритмов: SYN

Задача 1.

Доказать Утверждение 8.

Применение волновых алгоритмов: SYN

Задача 1.

Доказать Утверждение 8.

Задача 2.

Доказать Утверждение 9.

Применение волновых алгоритмов: INF

Волновые алгоритмы необходимы для вычисления некоторых функции, зависящих от входных данных каждого процесса. Примером такими функциями могут служить функции точной нижней грани по всем входам.

Применение волновых алгоритмов: INF

Волновые алгоритмы необходимы для вычисления некоторых функции, зависящих от входных данных каждого процесса. Примером такими функциями могут служить функции точной нижней грани по всем входам.

Пусть $(X, \leqslant)$ — частично упорядоченное множество. Тогда элемент c называется точной нижней гранью элементов a и b, если $c \leqslant a$, $c \leqslant b$ и $\forall d \ : \ (d \leqslant a \wedge d \leqslant b \implies d \leqslant c)$.
Если в $(X, \leqslant)$ точная нижняя грань всегда существует, то она определяется однозначно и обозначается $a \downarrow b$. Двуместная операция $\downarrow$ является коммутативной $(a \downarrow b = b \downarrow a)$ и ассоциативной $(a \downarrow (b \downarrow c) = (a \downarrow b) \downarrow c)$, и поэтому ее можно распространить на конечные множества:

$$\inf\{j_1, \ldots, j_k\} = j_1 \downarrow \cdots \downarrow j_k.$$

Применение волновых алгоритмов: INF

Волновые алгоритмы необходимы для вычисления некоторых функции, зависящих от входных данных каждого процесса. Примером такими функциями могут служить функции точной нижней грани по всем входам.

Пусть $(X, \leqslant)$ — частично упорядоченное множество. Тогда элемент c называется точной нижней гранью элементов a и b, если $c \leqslant a$, $c \leqslant b$ и $\forall d \ : \ (d \leqslant a \wedge d \leqslant b \implies d \leqslant c)$.
Если в $(X, \leqslant)$ точная нижняя грань всегда существует, то она определяется однозначно и обозначается $a \downarrow b$. Двуместная операция $\downarrow$ является коммутативной $(a \downarrow b = b \downarrow a)$ и ассоциативной $(a \downarrow (b \downarrow c) = (a \downarrow b) \downarrow c)$, и поэтому ее можно распространить на конечные множества:

$$\inf\{j_1, \ldots, j_k\} = j_1 \downarrow \cdots \downarrow j_k.$$

Примерами функции $\downarrow$ могут служить операции $min, max, \wedge, \vee, \cap, \cup$.

Применение волновых алгоритмов: INF

Проблема вычисления точной нижней грани (Infimum Computation, INF) такова. Все процессы q наделены входными данными d_q, которые являются элементами частично упорядоченного множества X. Требуется, чтобы некоторые выделенные процессы вычислили значение $\downarrow \{d_p \ : \ q \in \mathbb{P}\}$ и чтобы эти процессы были осведомлены о завершении вычисления. Они записывают результат вычисления в переменной *out*, и им не разрешается впоследствии изменять значение этой переменной.

Запись значения в переменную *out*, рассматривается как событие *decide* в INF-алгоритме.

Применение волновых алгоритмов: INF

Проблема вычисления точной нижней грани (Infimum Computation, INF) такова. Все процессы q наделены входными данными d_q, которые являются элементами частично упорядоченного множества X. Требуется, чтобы некоторые выделенные процессы вычислили значение $\downarrow \{d_p \ : \ q \in \mathbb{P}\}$ и чтобы эти процессы были осведомлены о завершении вычисления. Они записывают результат вычисления в переменной *out*, и им не разрешается впоследствии изменять значение этой переменной.

Запись значения в переменную *out*, рассматривается как событие *decide* в INF-алгоритме.

Утверждение 10.

Каждый волновой алгоритм может быть использован в качестве INF-алгоритма.

Применение волновых алгоритмов: INF

Проблема вычисления точной нижней грани (Infimum Computation, INF) такова. Все процессы q наделены входными данными d_q, которые являются элементами частично упорядоченного множества X. Требуется, чтобы некоторые выделенные процессы вычислили значение $\downarrow \{d_p \ : \ q \in \mathbb{P}\}$ и чтобы эти процессы были осведомлены о завершении вычисления. Они записывают результат вычисления в переменной *out*, и им не разрешается впоследствии изменять значение этой переменной.

Запись значения в переменную *out*, рассматривается как событие *decide* в INF-алгоритме.

Утверждение 10.

Каждый волновой алгоритм может быть использован в качестве INF-алгоритма.

Утверждение 11.

Каждый INF-алгоритм может быть использован в качестве волнового алгоритма.

Применение волновых алгоритмов: SYN

Задача 3.

Доказать Утверждение 10.

Применение волновых алгоритмов: SYN

Задача 3.

Доказать Утверждение 10.

Задача 4.

Доказать Утверждение 11.

Примеры волновых алгоритмов

Кольцевой алгоритм

Предназначен для кольцевой сети и для гамильтоновой сети, в которой информация о некотором фиксированном гамильтоновом цикле заложена во всех процессах.

Предполагается, что каждому процессу p предписан сосед $Next_p$ так, чтобы каналы связи между выделенными таким образом соседями образовывали гамильтонов цикл.

Кольцевой алгоритм

Алгоритм является централизованным: инициатор отправляет сообщение **tok** (оно называется маркером) по циклу, каждый процесс передает его далее, и, когда оно возвращается инициатору, тот принимает решение

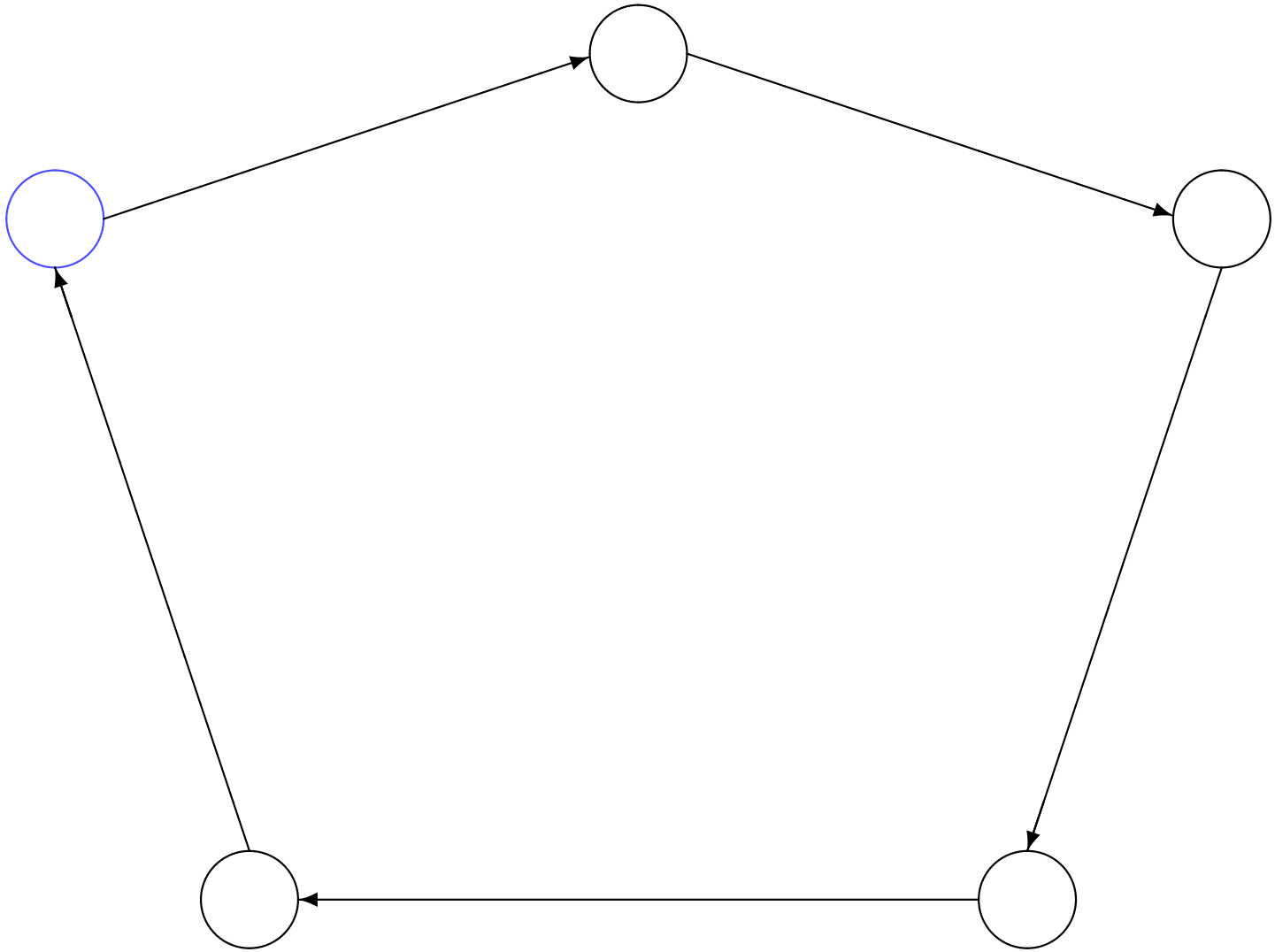
Для инициатора:

begin send **tok** to $Next_p$; receive **tok** ; decide **end**

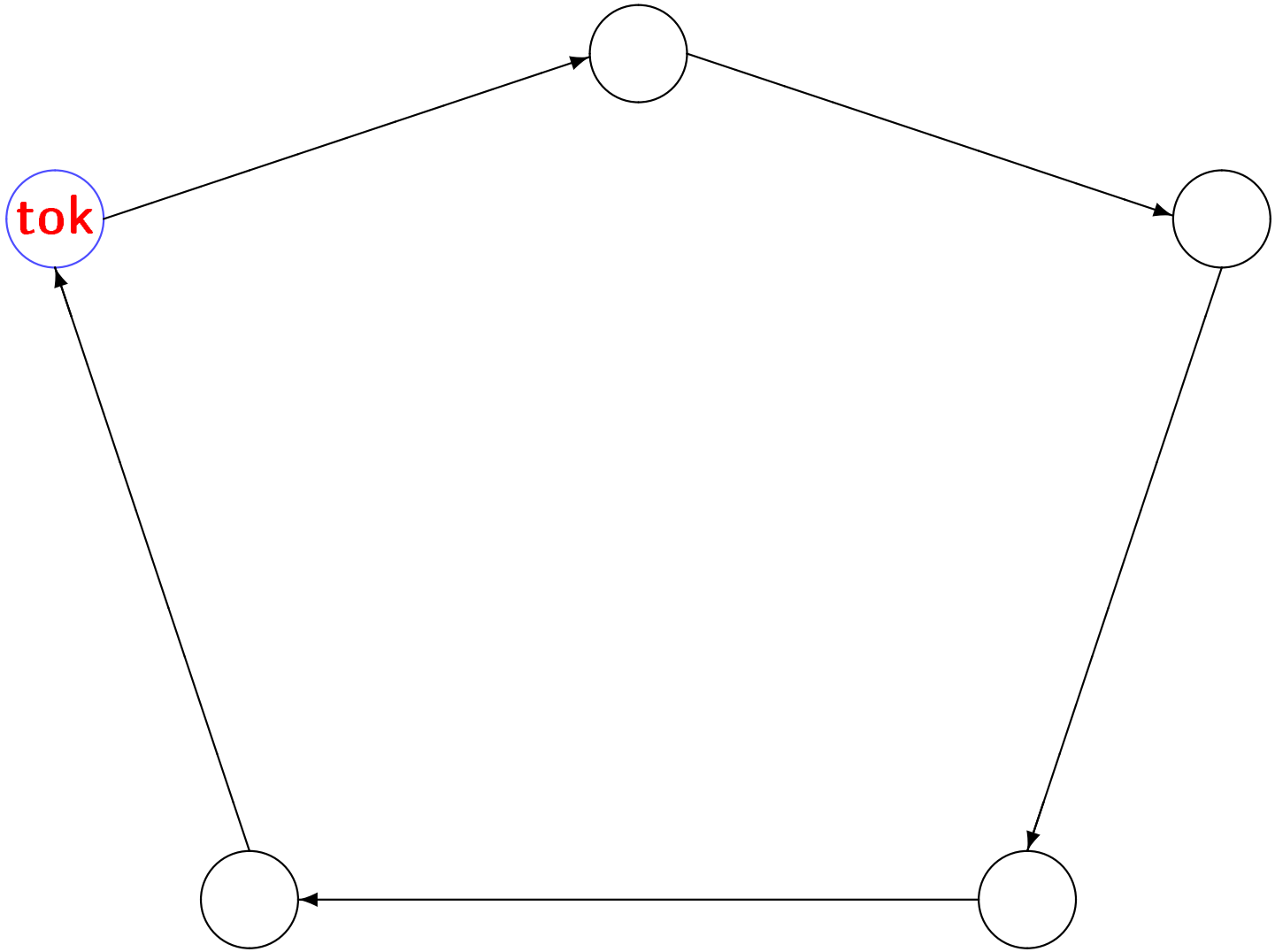
Для неинициатора:

begin receive **tok**; send **tok** to $Next_p$ **end**

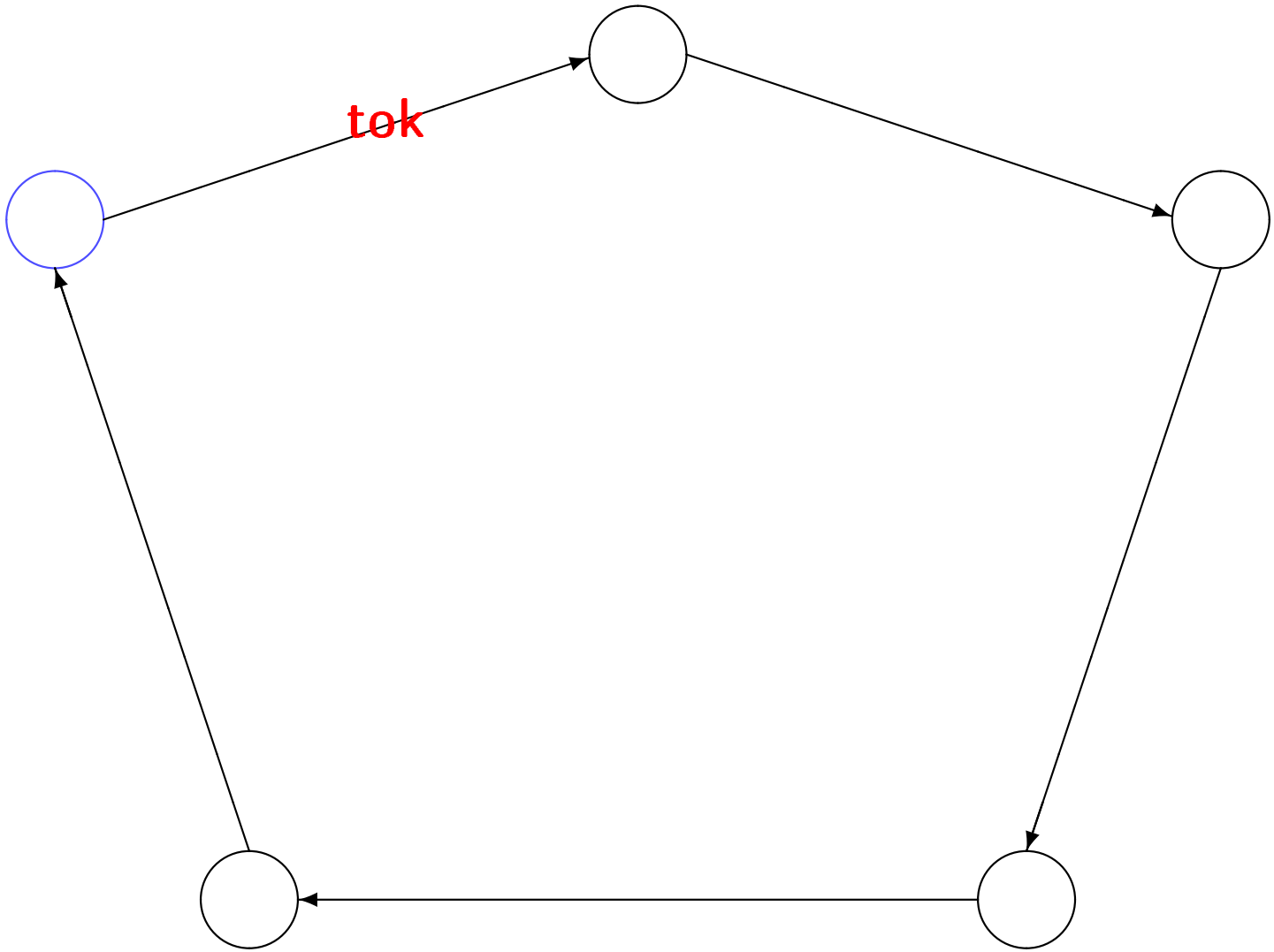
Кольцевой алгоритм



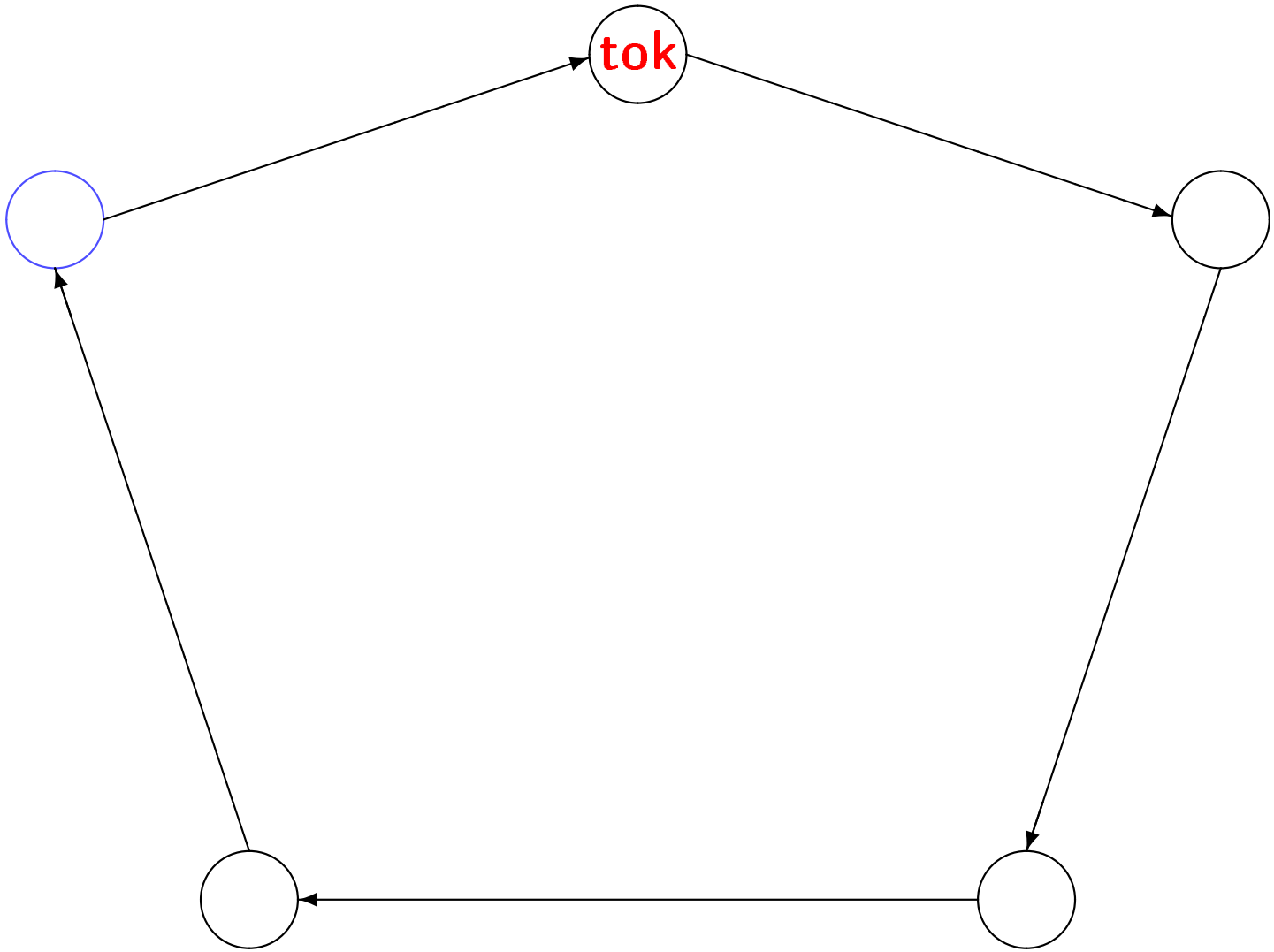
Кольцевой алгоритм



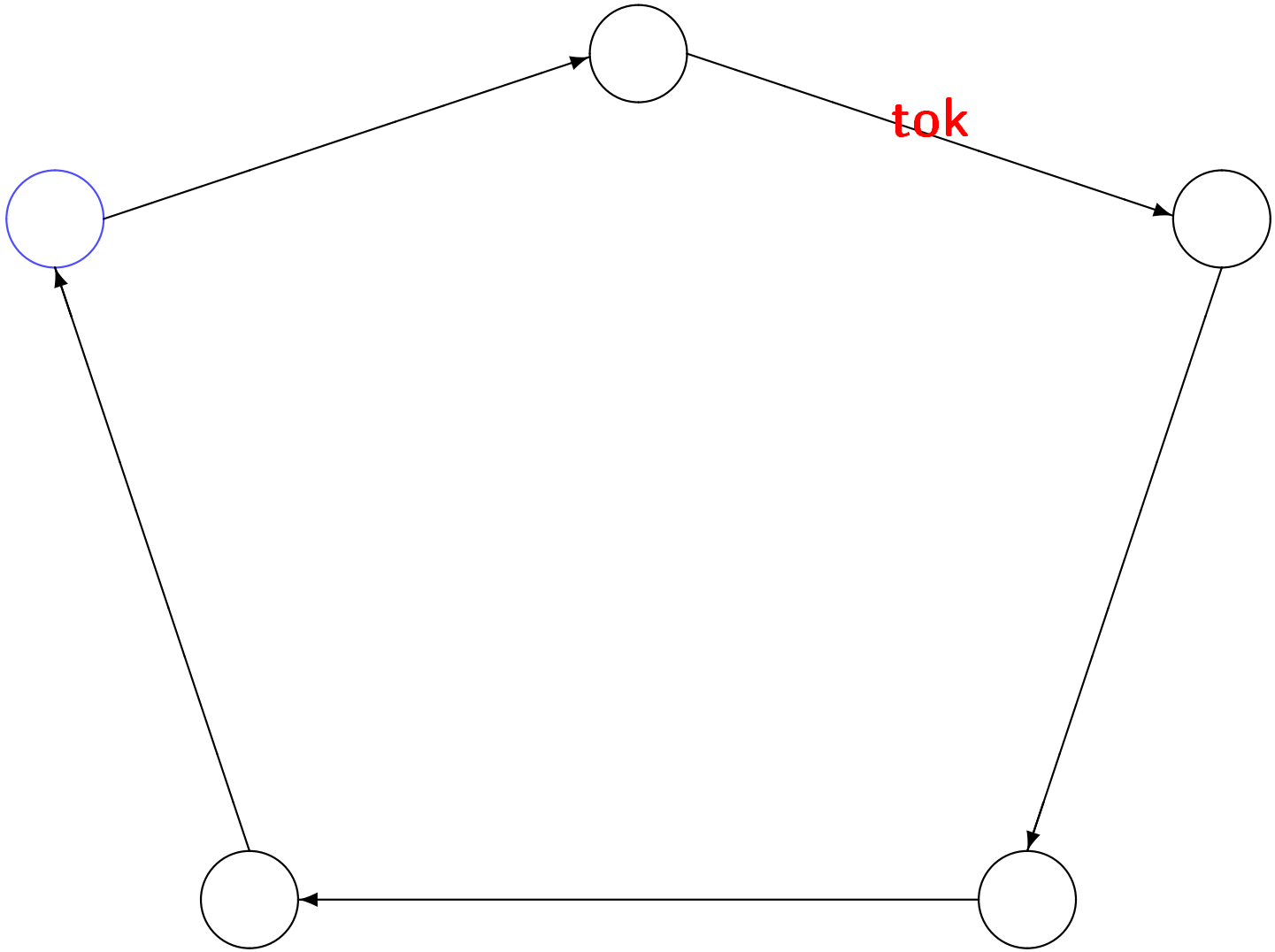
Кольцевой алгоритм



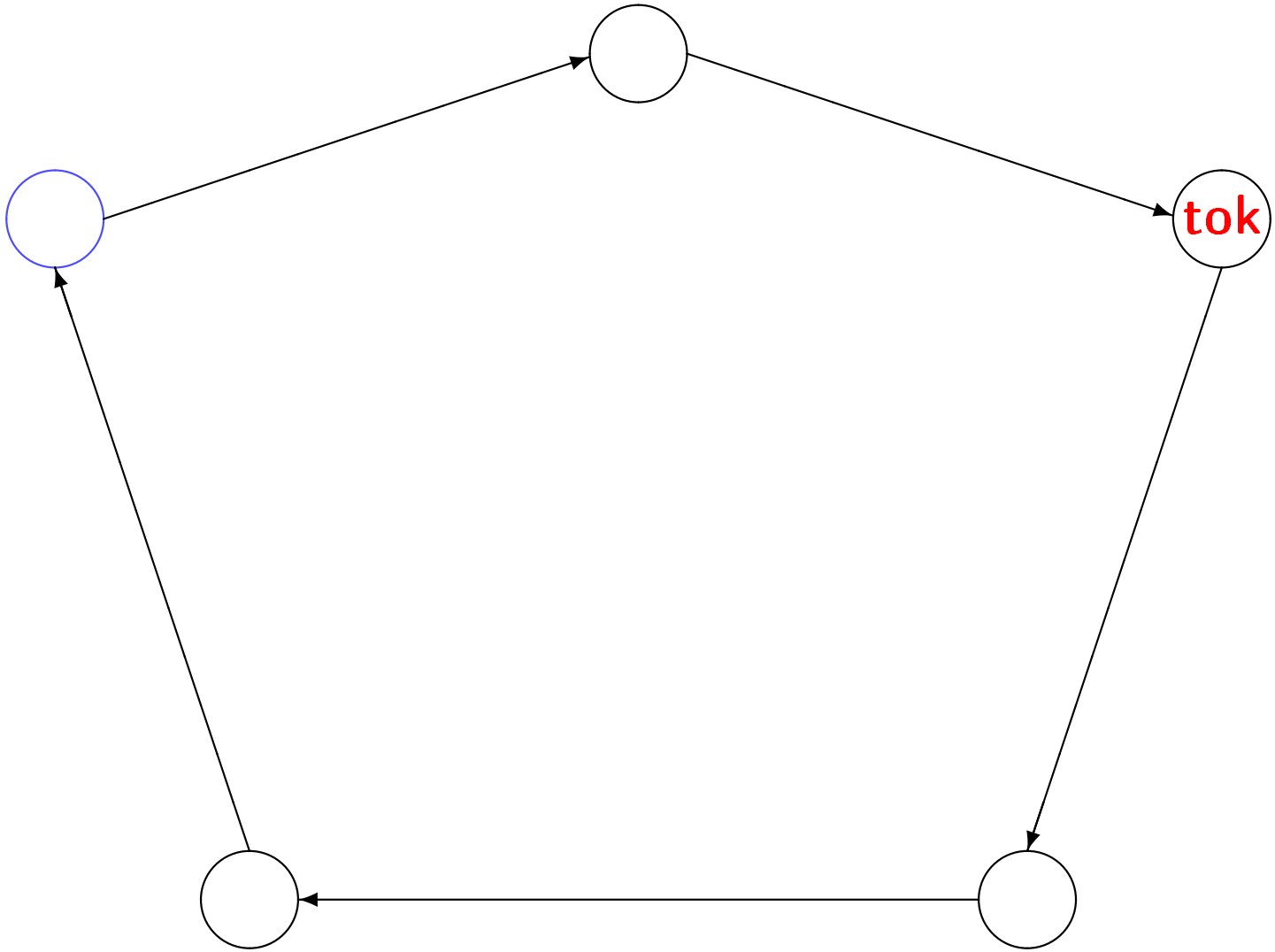
Кольцевой алгоритм



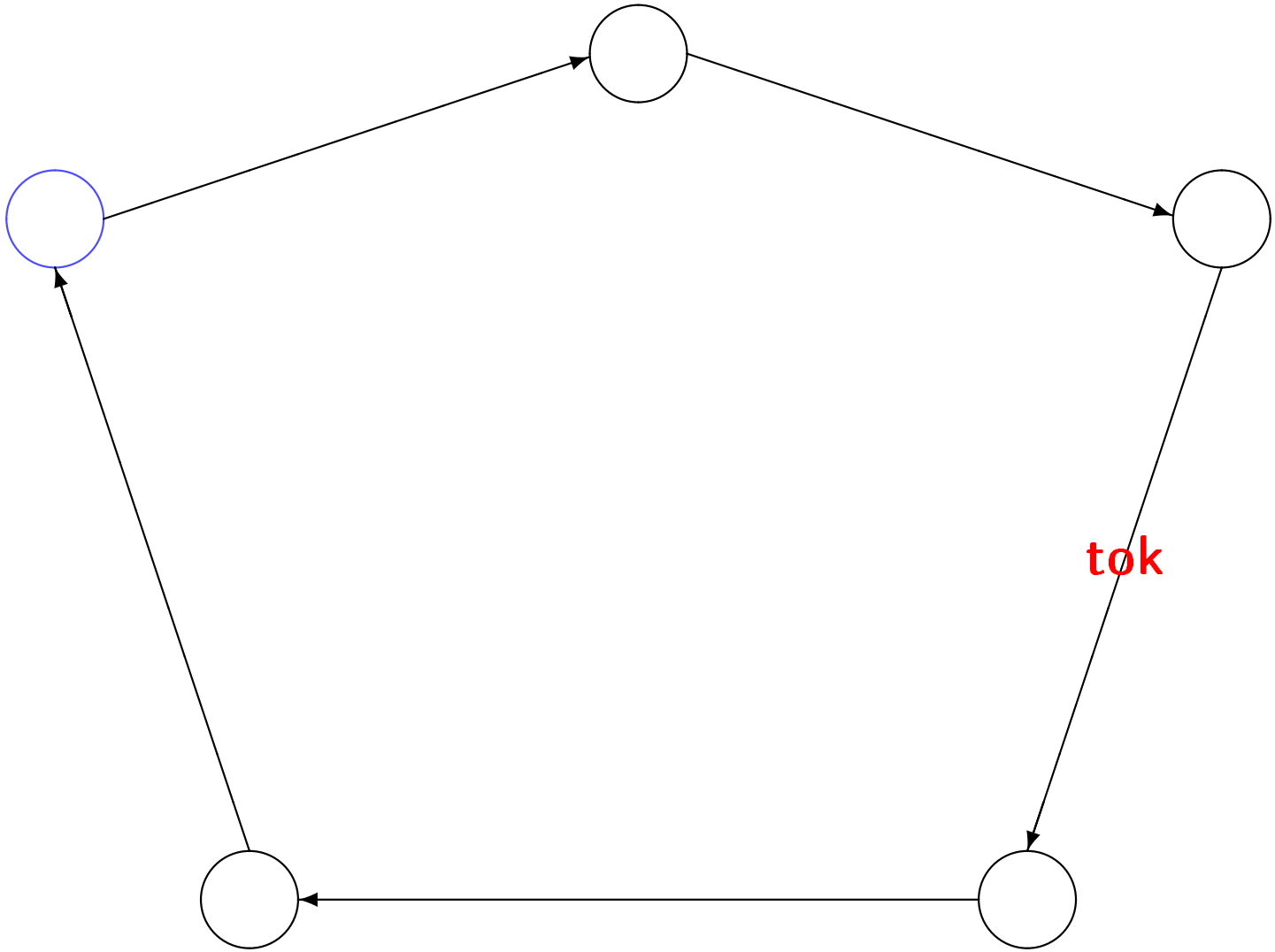
Кольцевой алгоритм



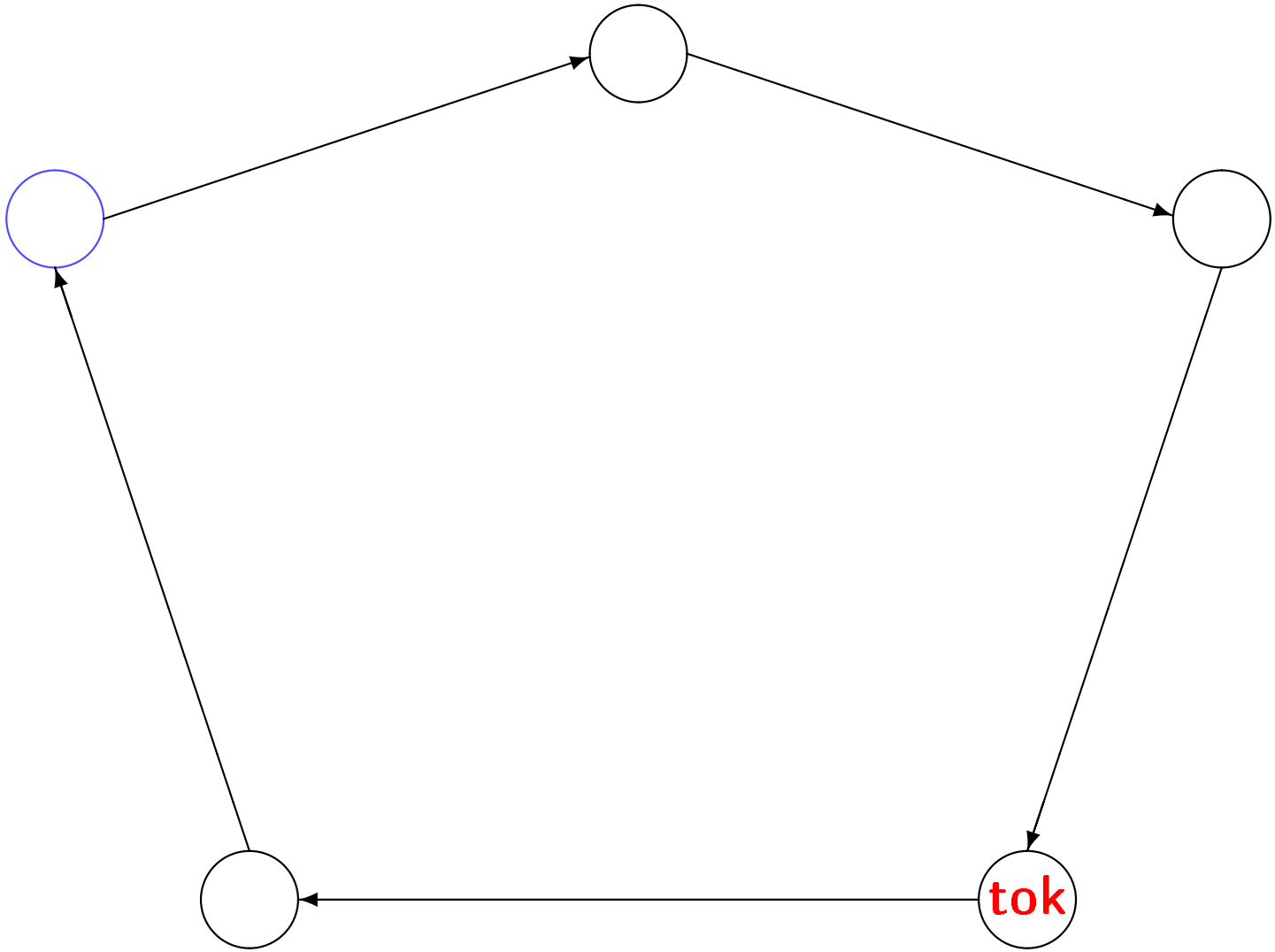
Кольцевой алгоритм



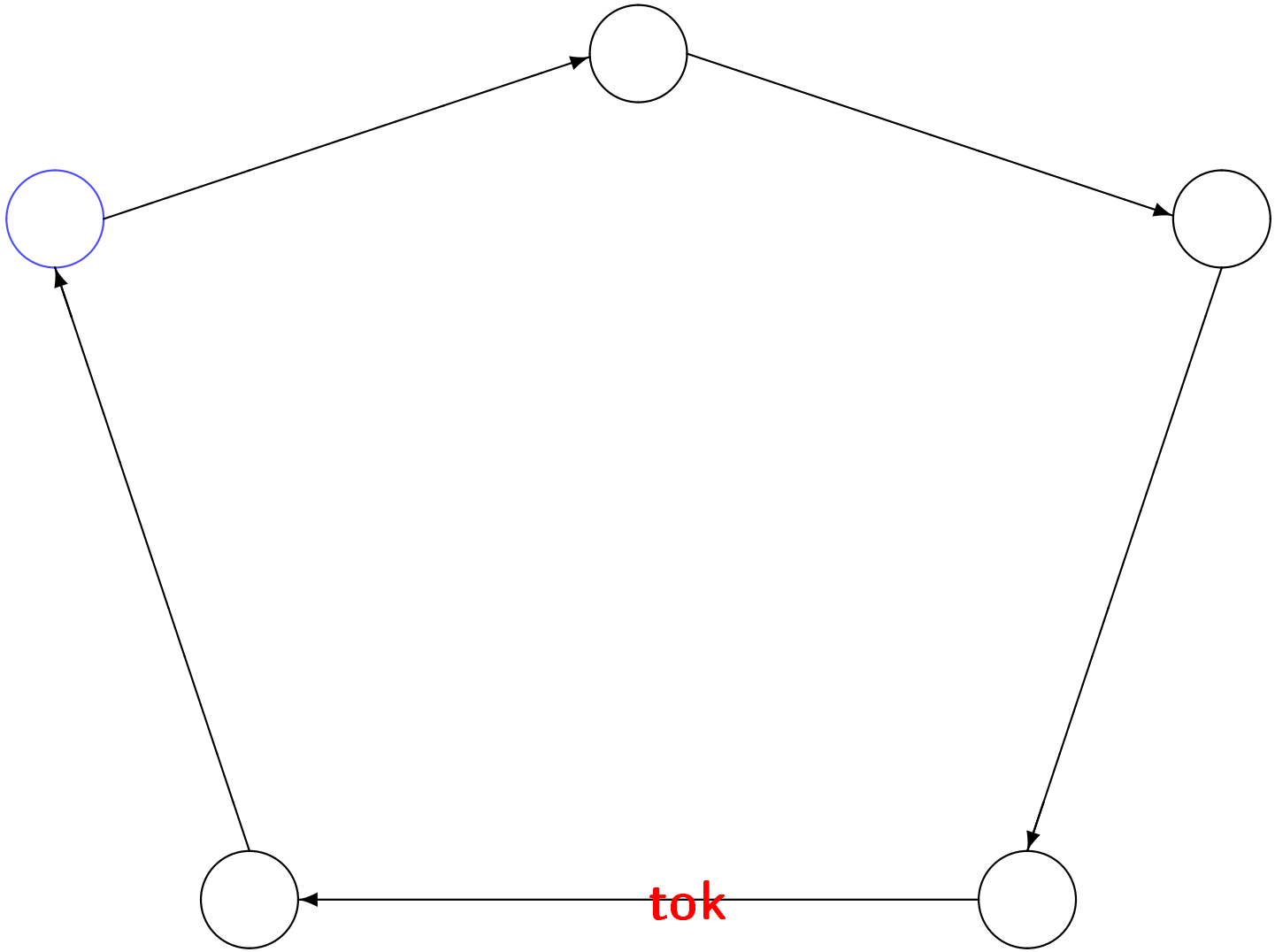
Кольцевой алгоритм



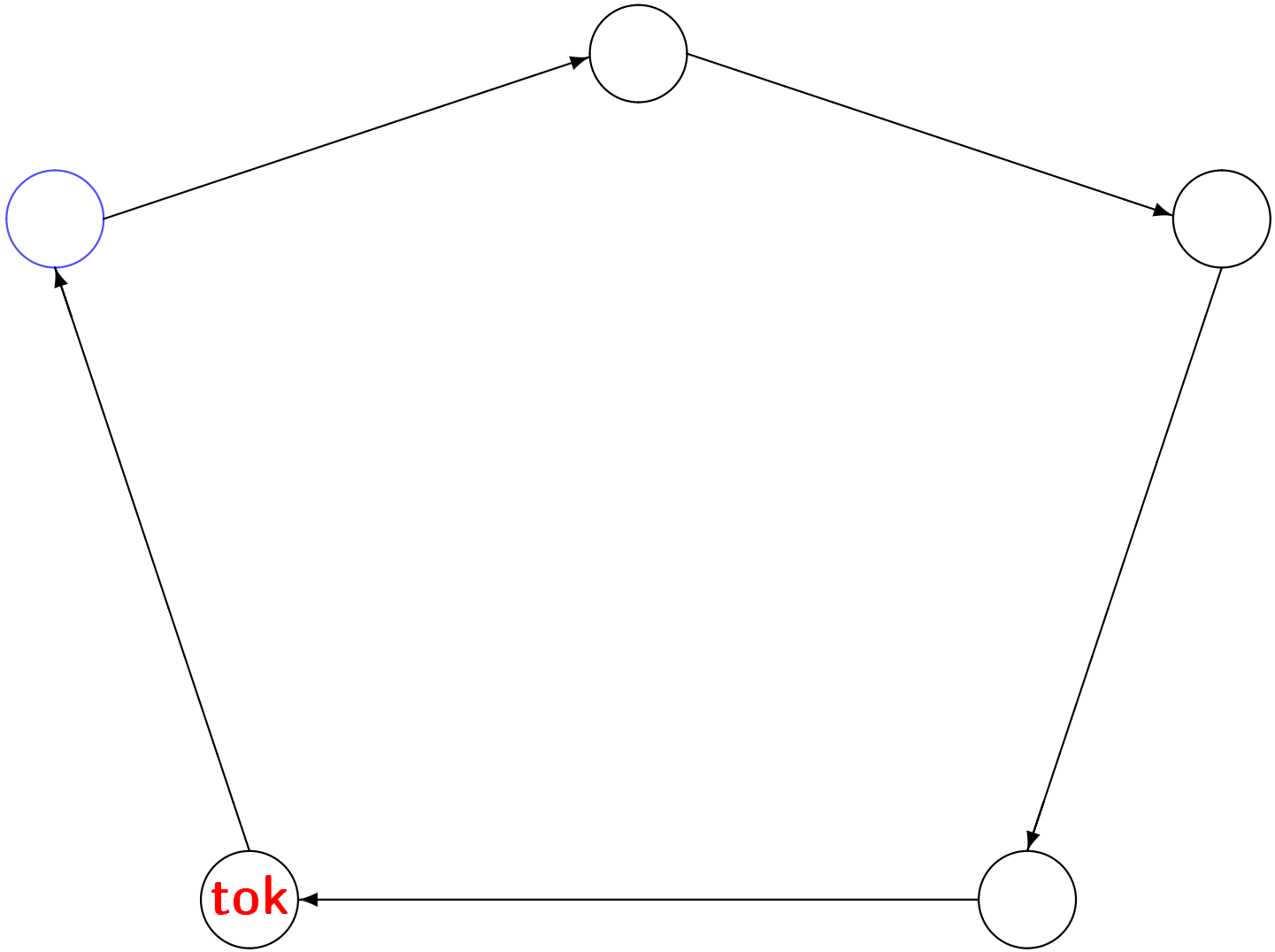
Кольцевой алгоритм



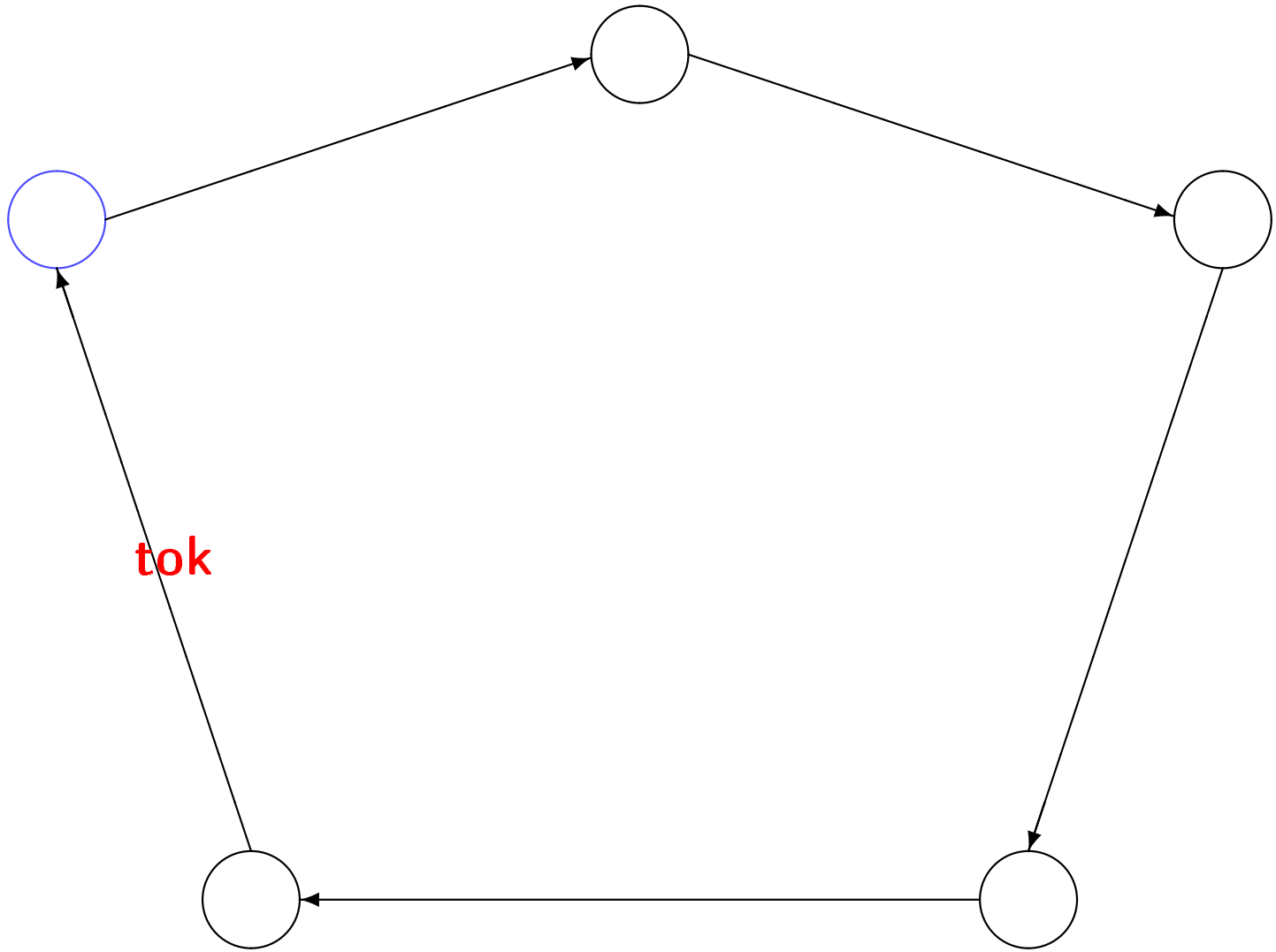
Кольцевой алгоритм



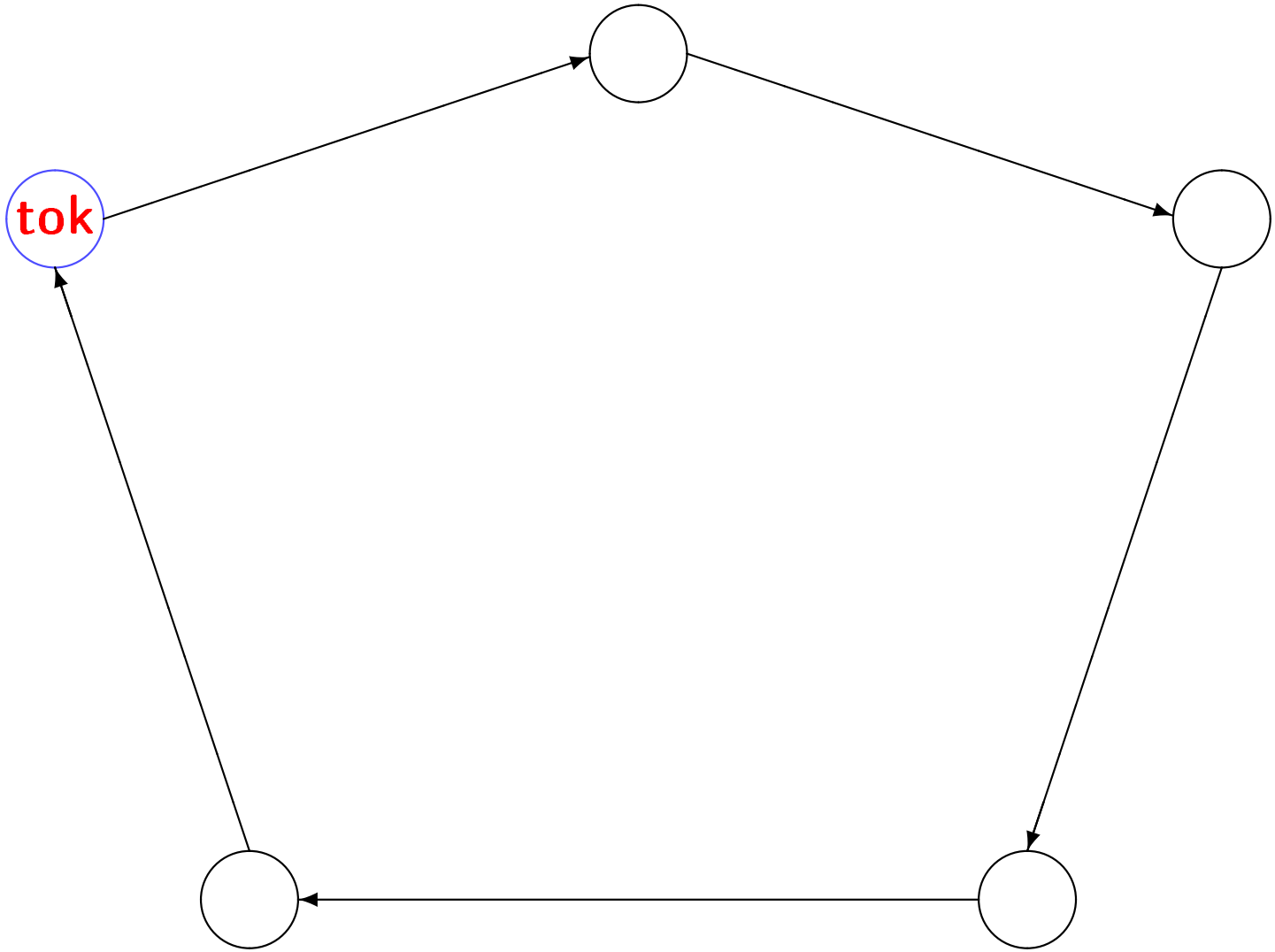
Кольцевой алгоритм



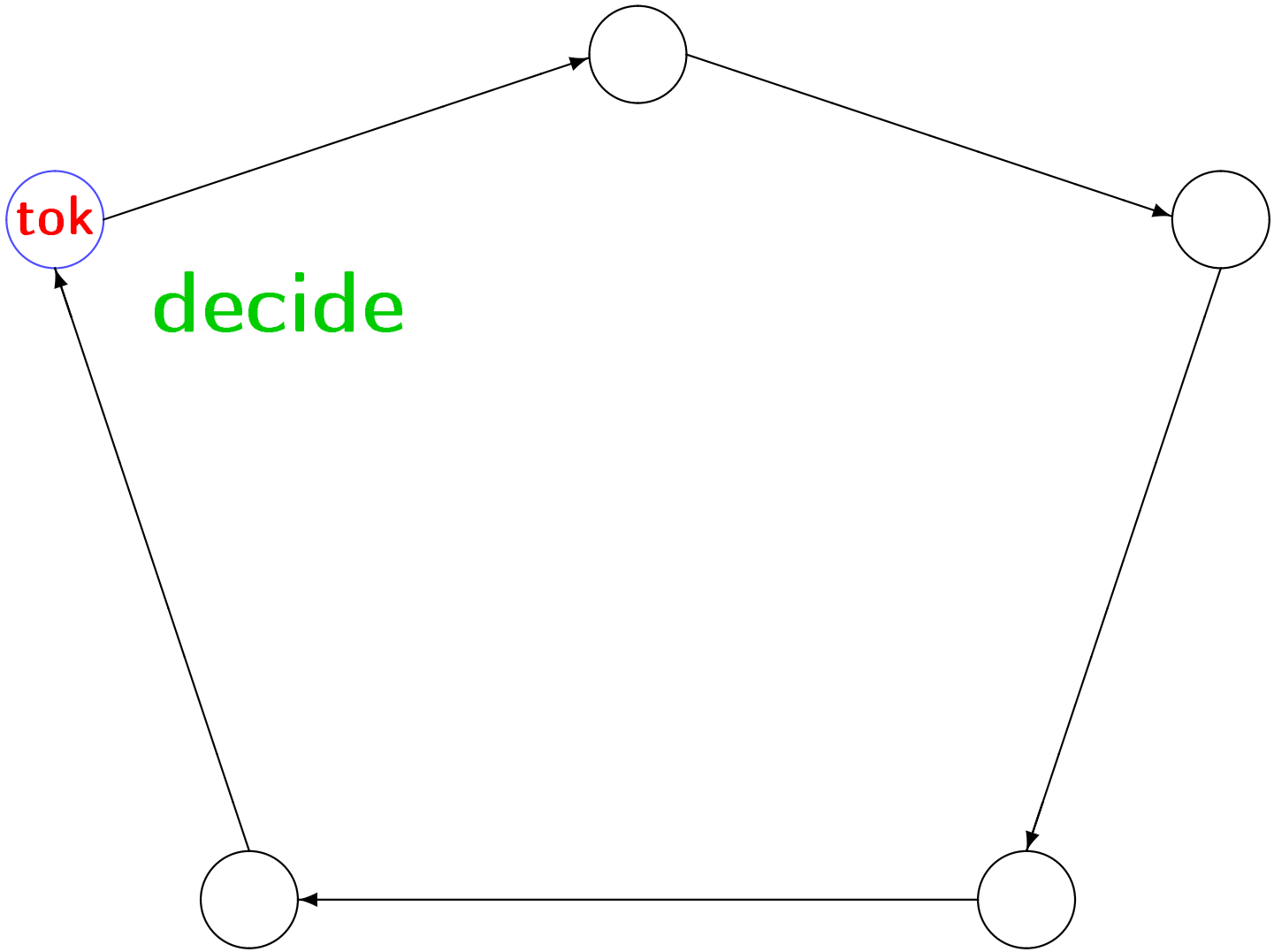
Кольцевой алгоритм



Кольцевой алгоритм



Кольцевой алгоритм



Кольцевой алгоритм

Утверждение 12.

Кольцевой алгоритм является волновым алгоритмом.

Кольцевой алгоритм

Утверждение 12.

Кольцевой алгоритм является волновым алгоритмом.

Задача 5.

Доказать Утверждение 12.

Кольцевой алгоритм

Утверждение 12.

Кольцевой алгоритм является волновым алгоритмом.

Задача 5.

Доказать Утверждение 12.

Задача 6.

А могут ли волновые алгоритмы вычислять суммы?

Примеры волновых алгоритмов

Древесный алгоритм

Предназначен для древесной сети, но может быть использован и для произвольной сети, имеющей доступ к остовному дереву этой сети.

Инициаторы — все листовые вершины дерева.

Идея.

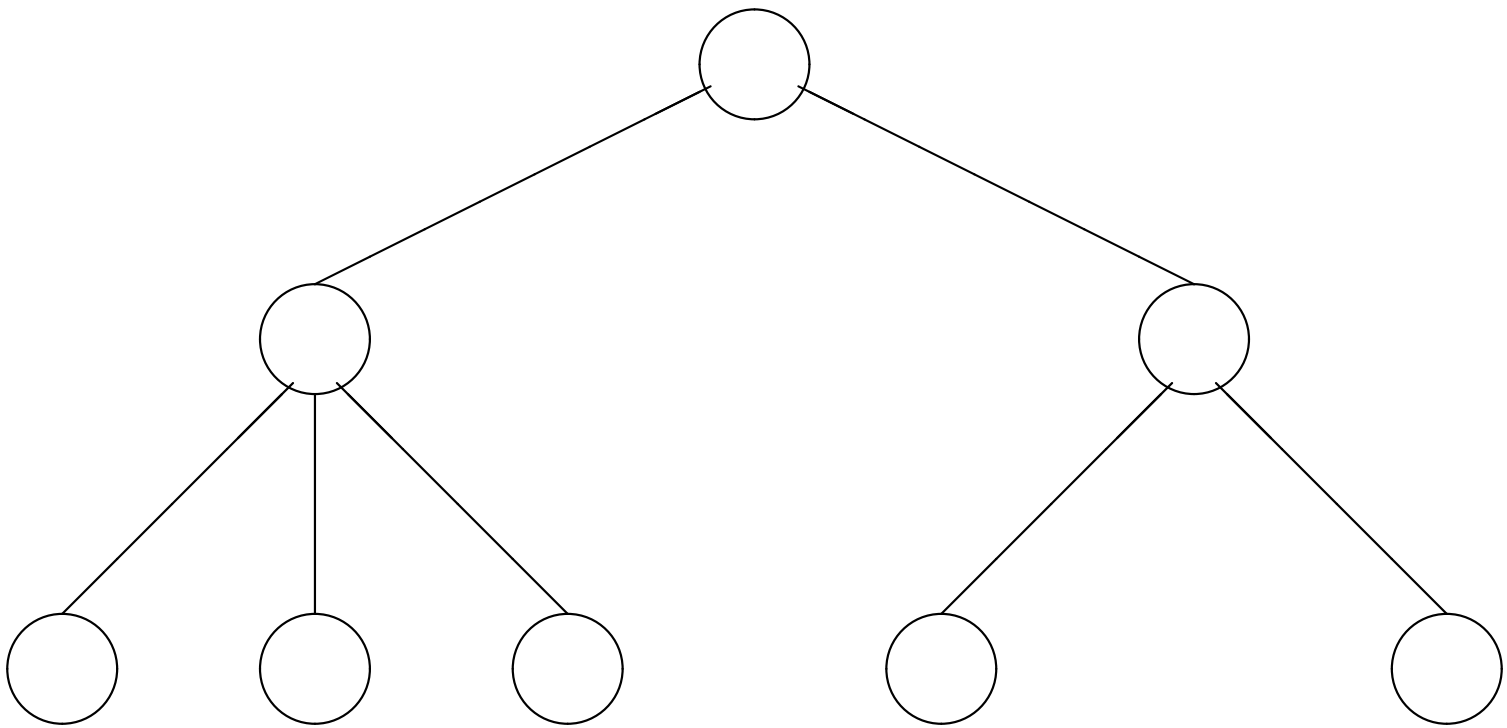
Все листья сети запускают алгоритм.

Каждый процесс отправляет в точности одно сообщение на протяжении работы алгоритма.

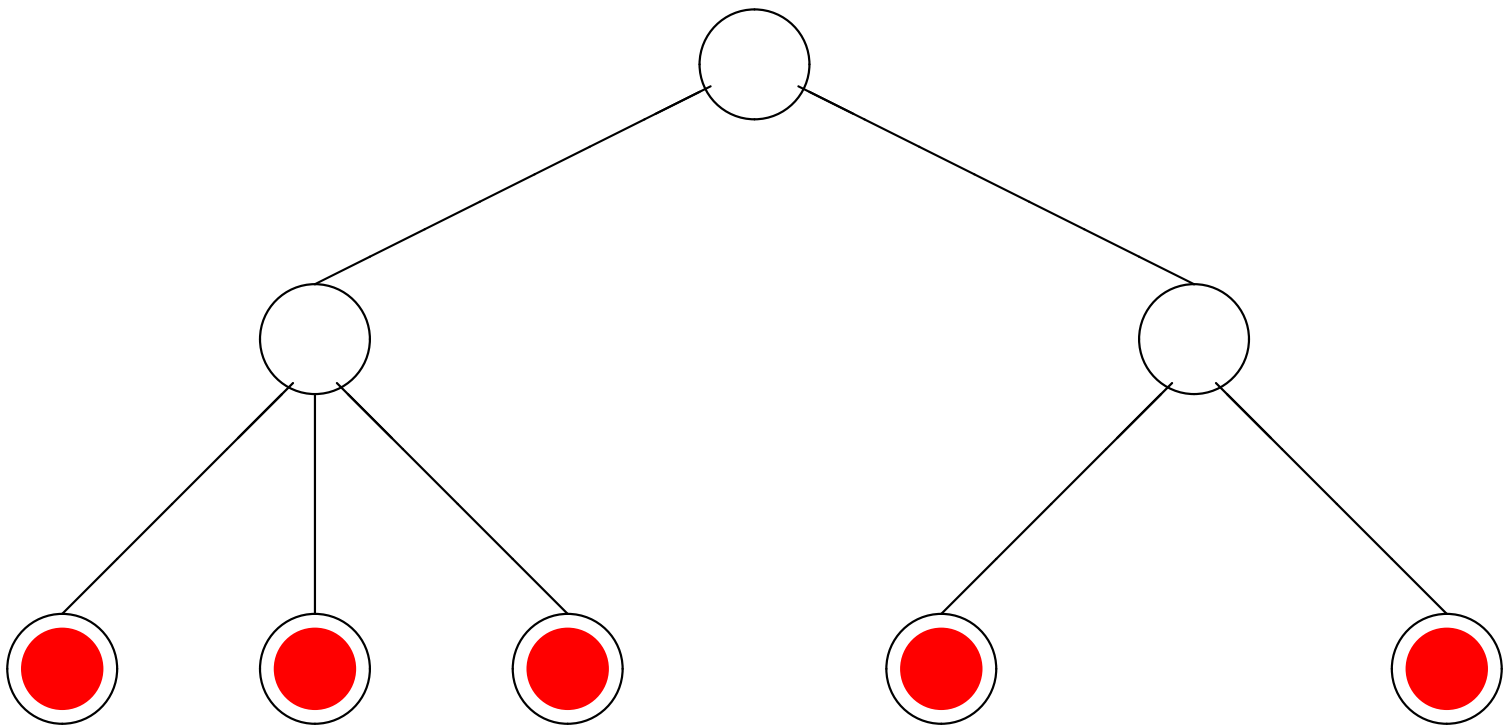
Если процесс уже получил сообщение по каждому из инцидентных ему каналов, кроме одного (это условие первоначально верно для листьев), то он отправляет сообщение по оставшемуся каналу.

Если процесс уже получил сообщение по всем инцидентным ему каналам, то он принимает решение.

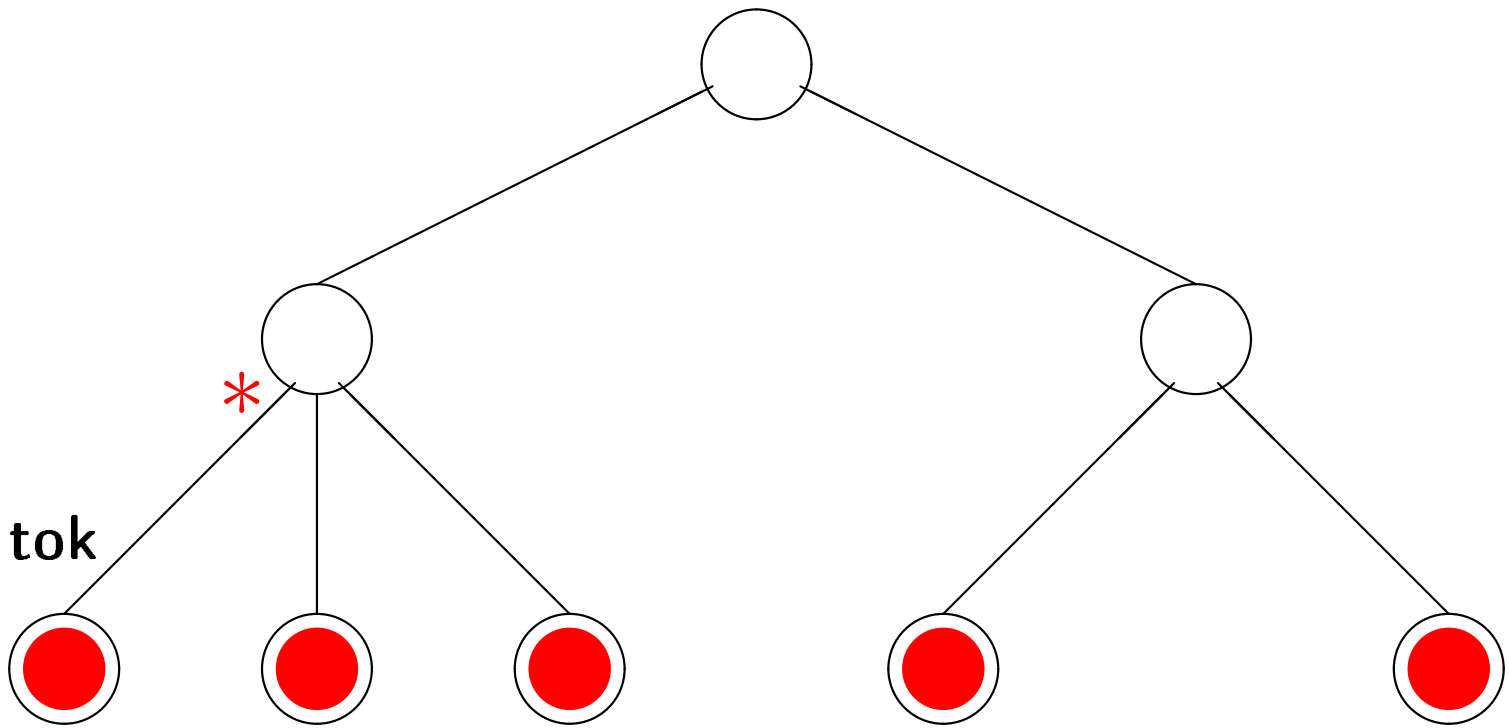
Древесный алгоритм



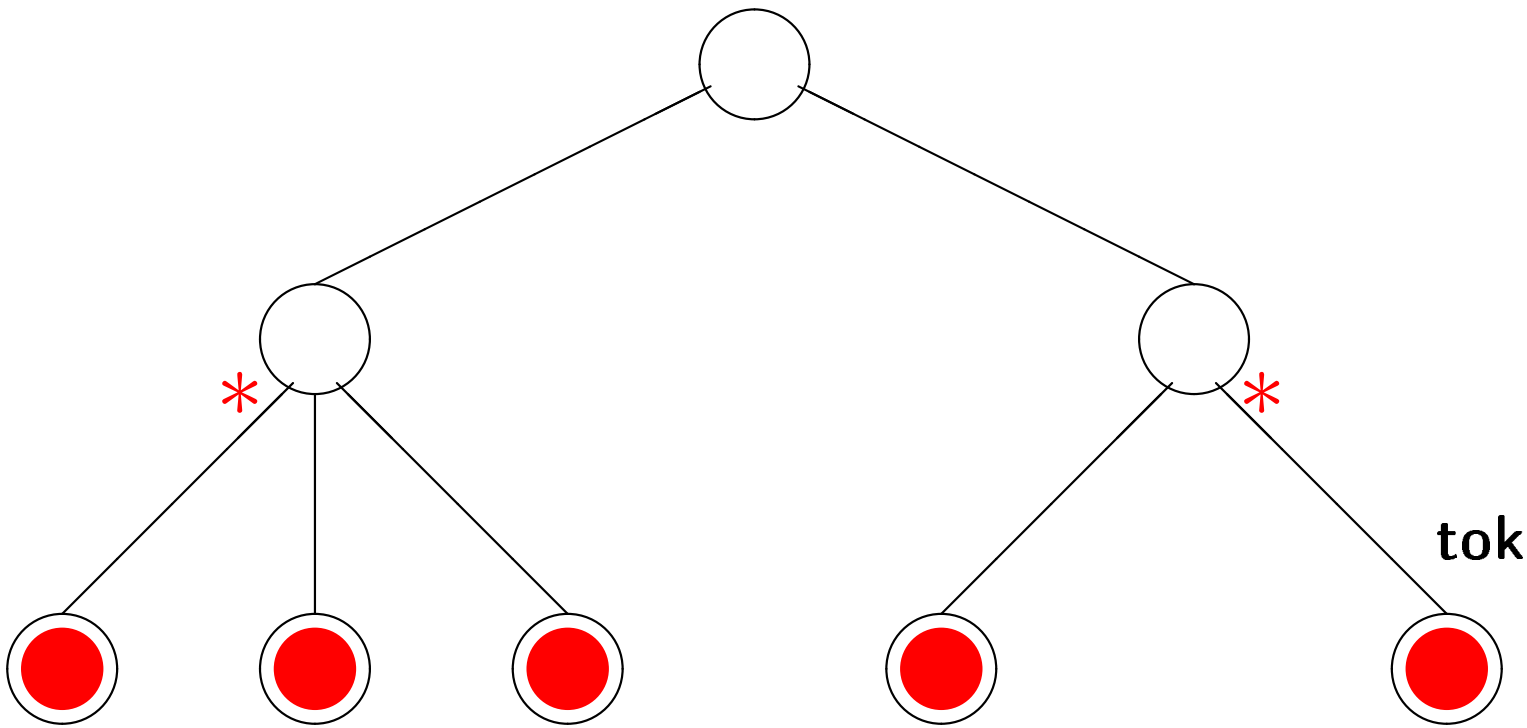
Древесный алгоритм



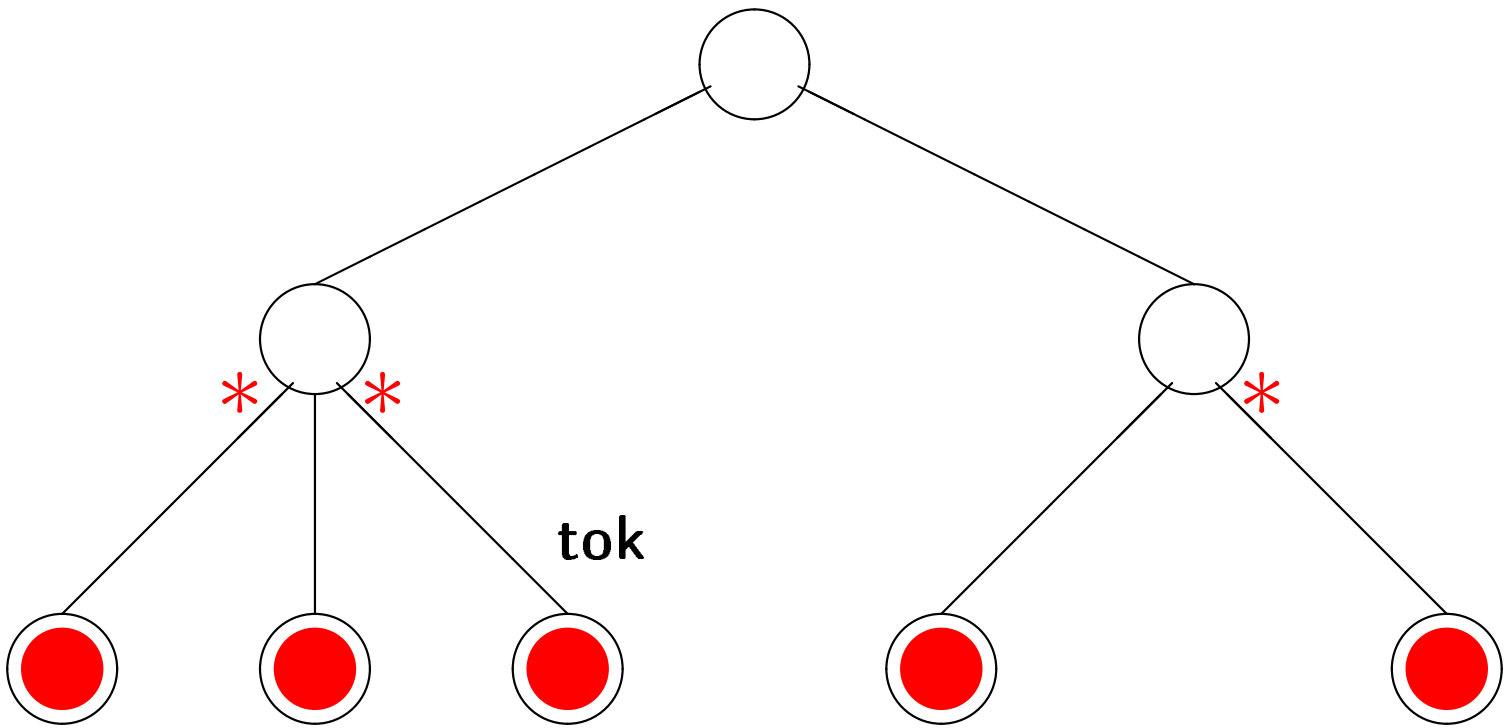
Древесный алгоритм



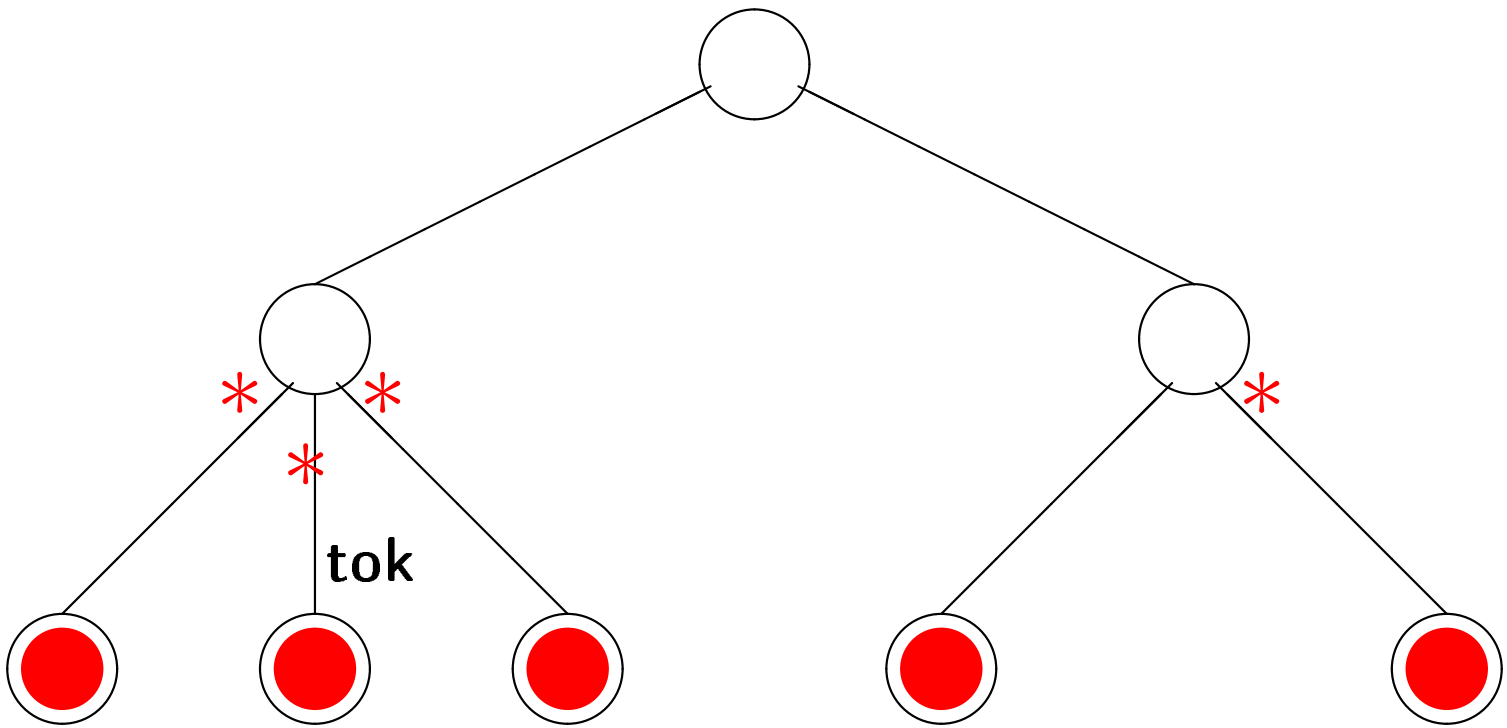
Древесный алгоритм



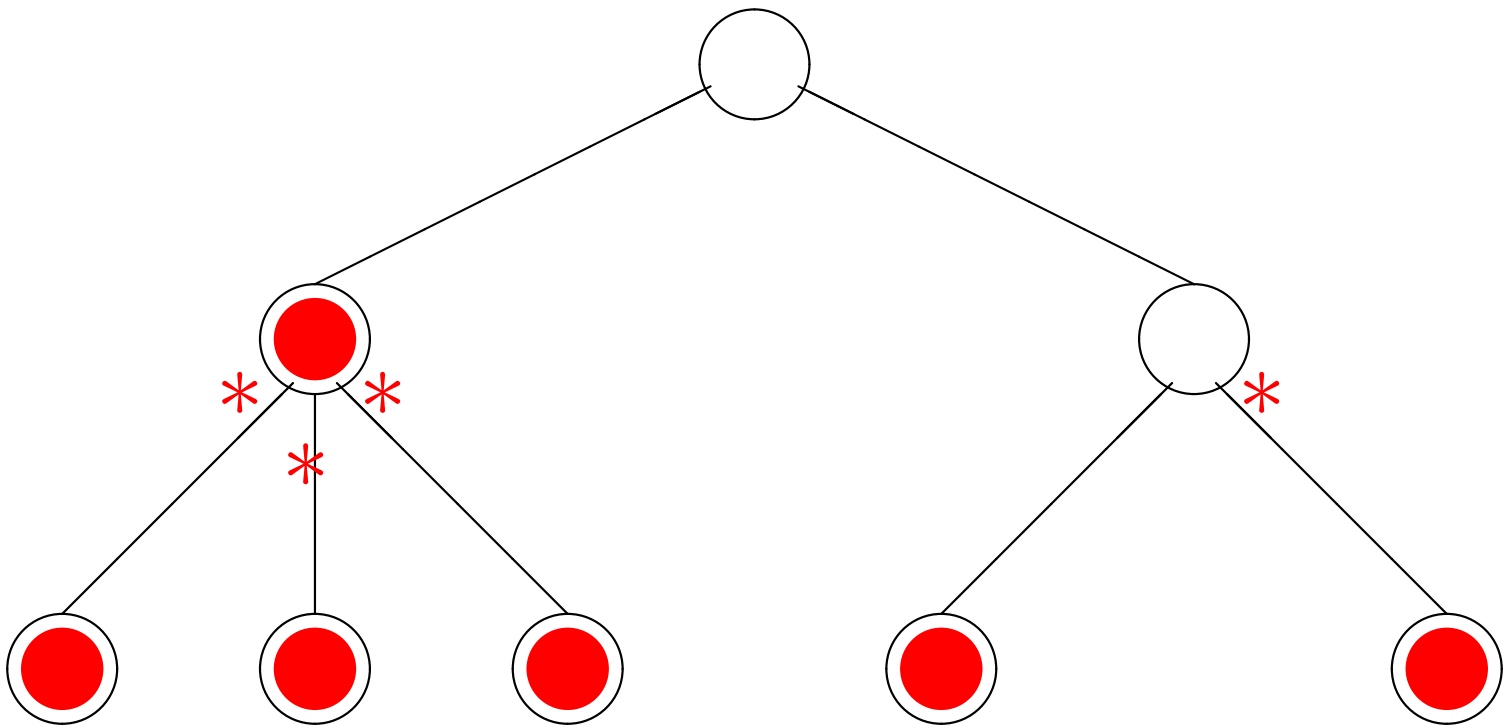
Древесный алгоритм



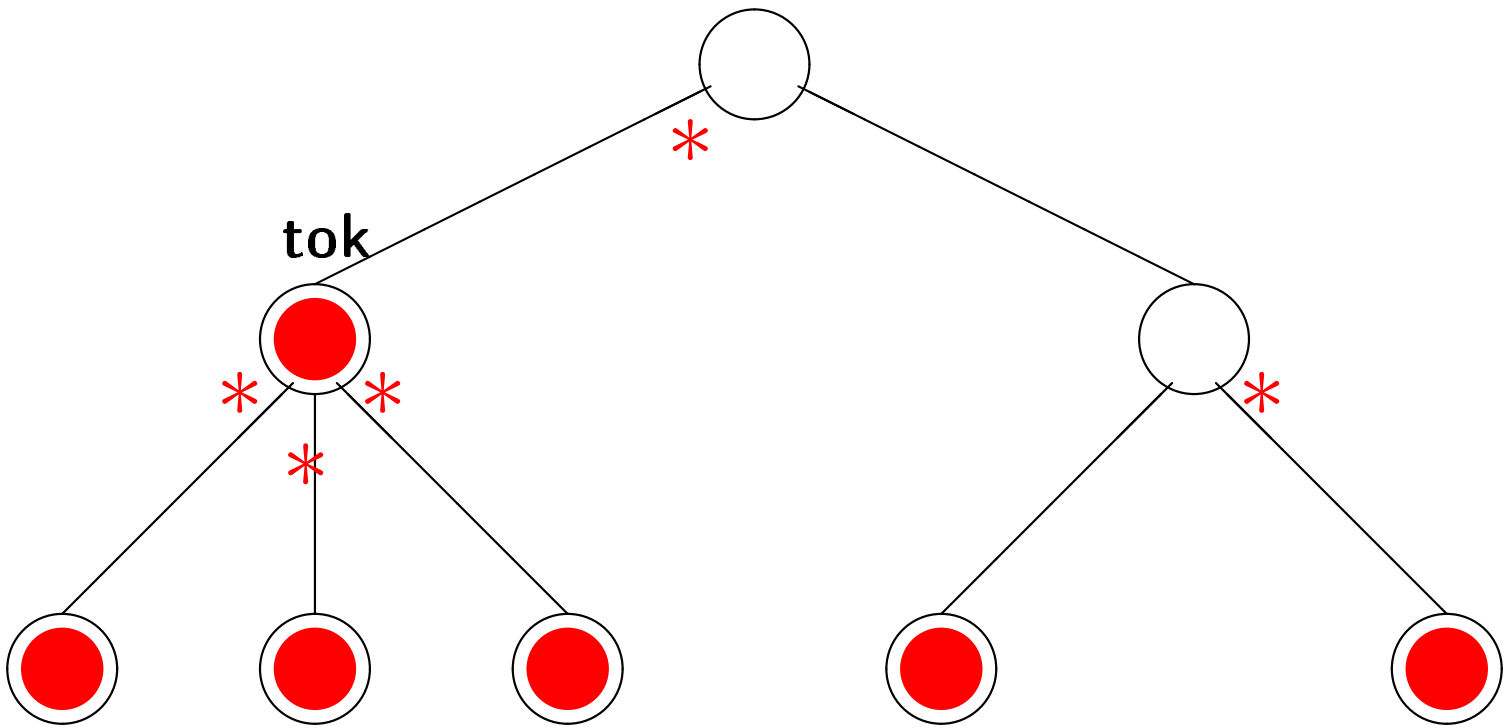
Древесный алгоритм



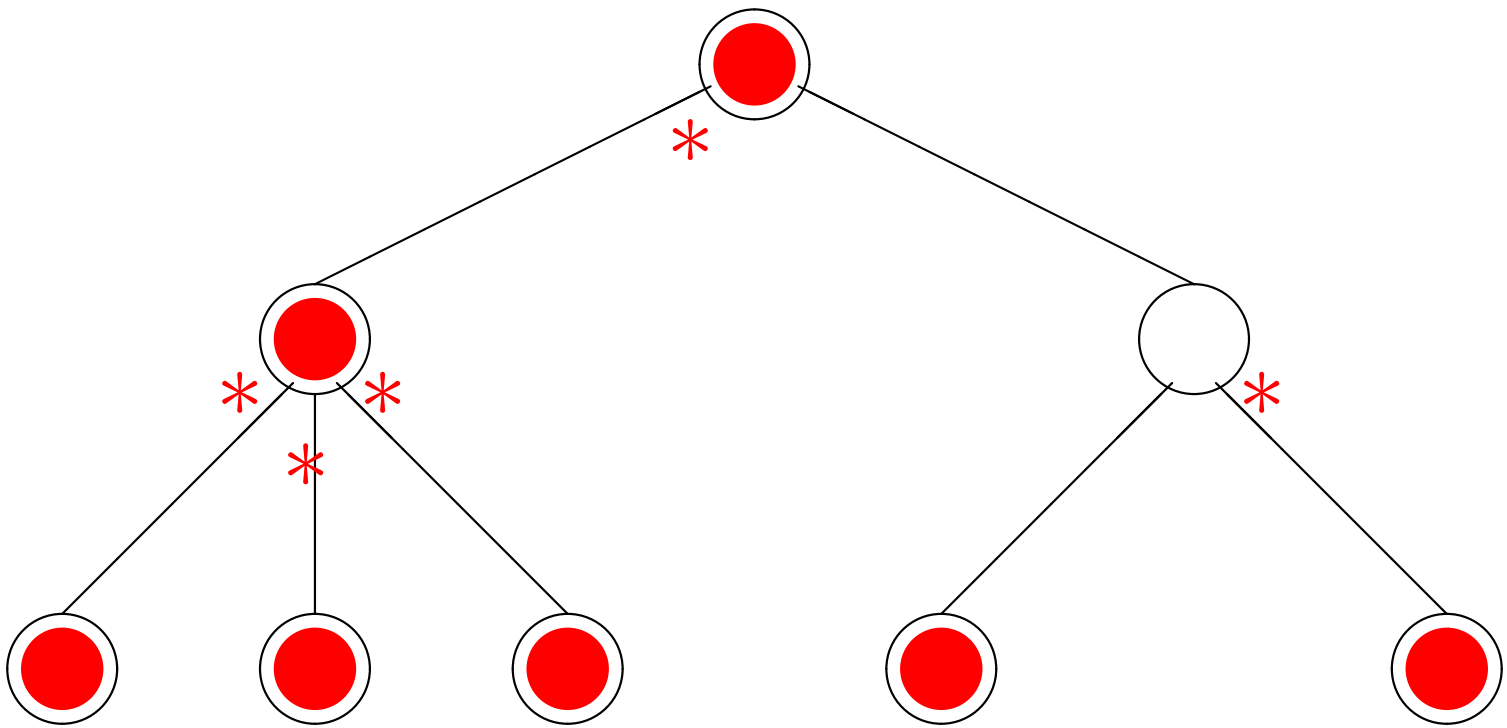
Древесный алгоритм



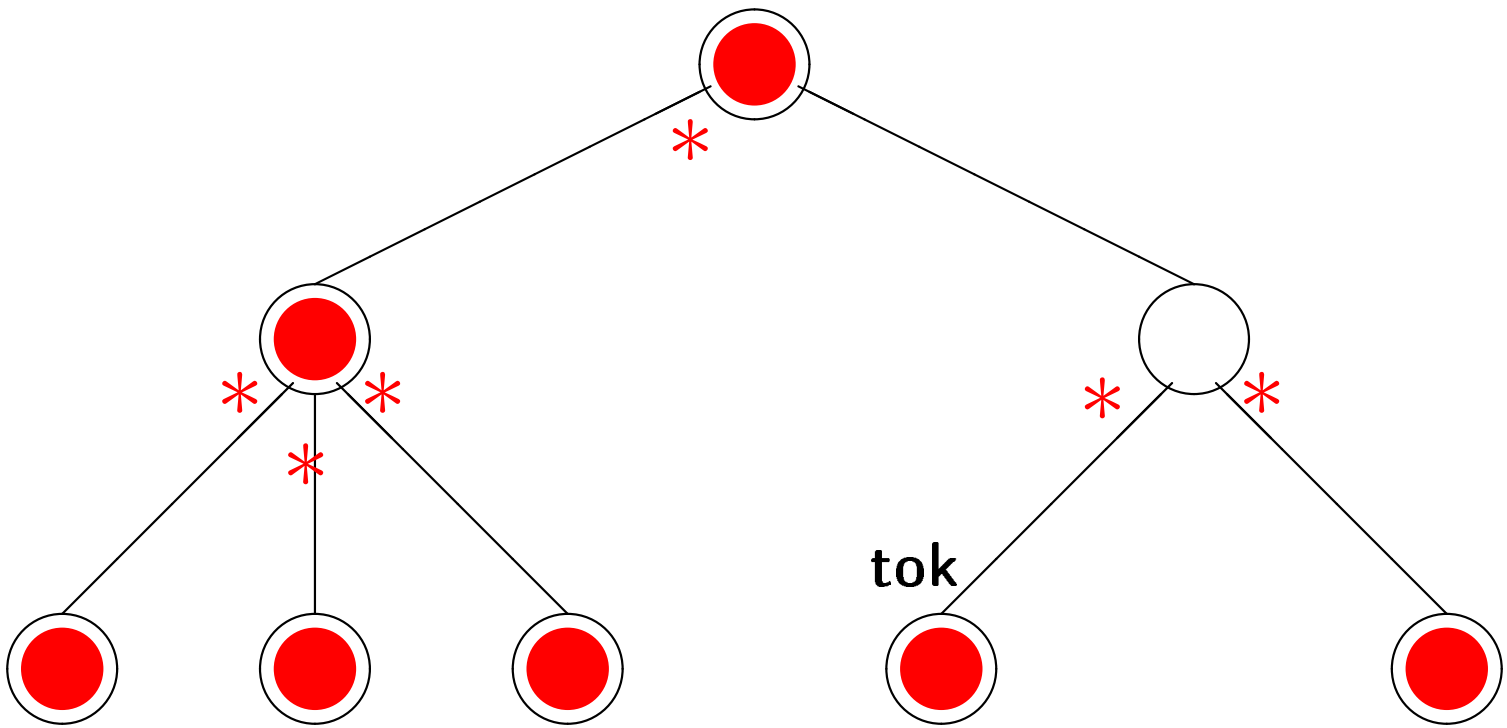
Древесный алгоритм



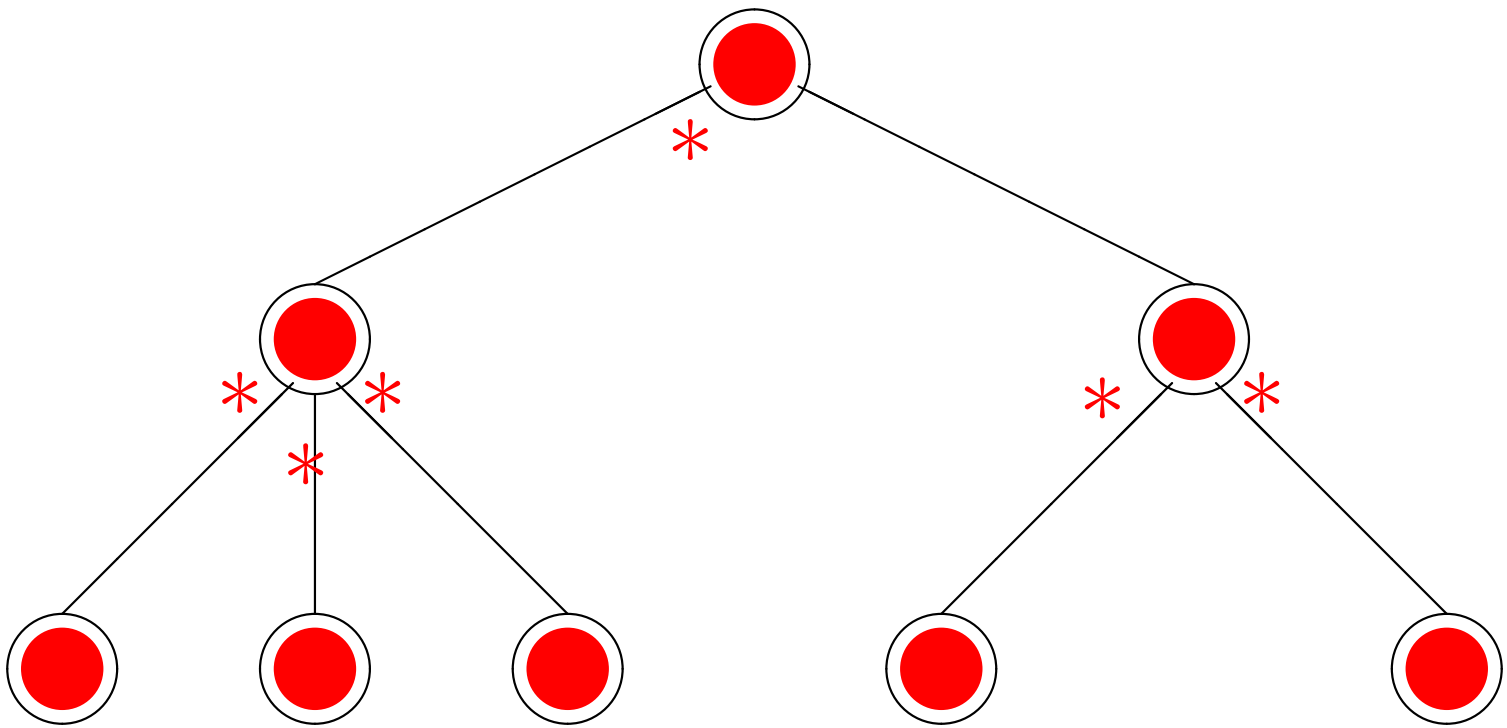
Древесный алгоритм



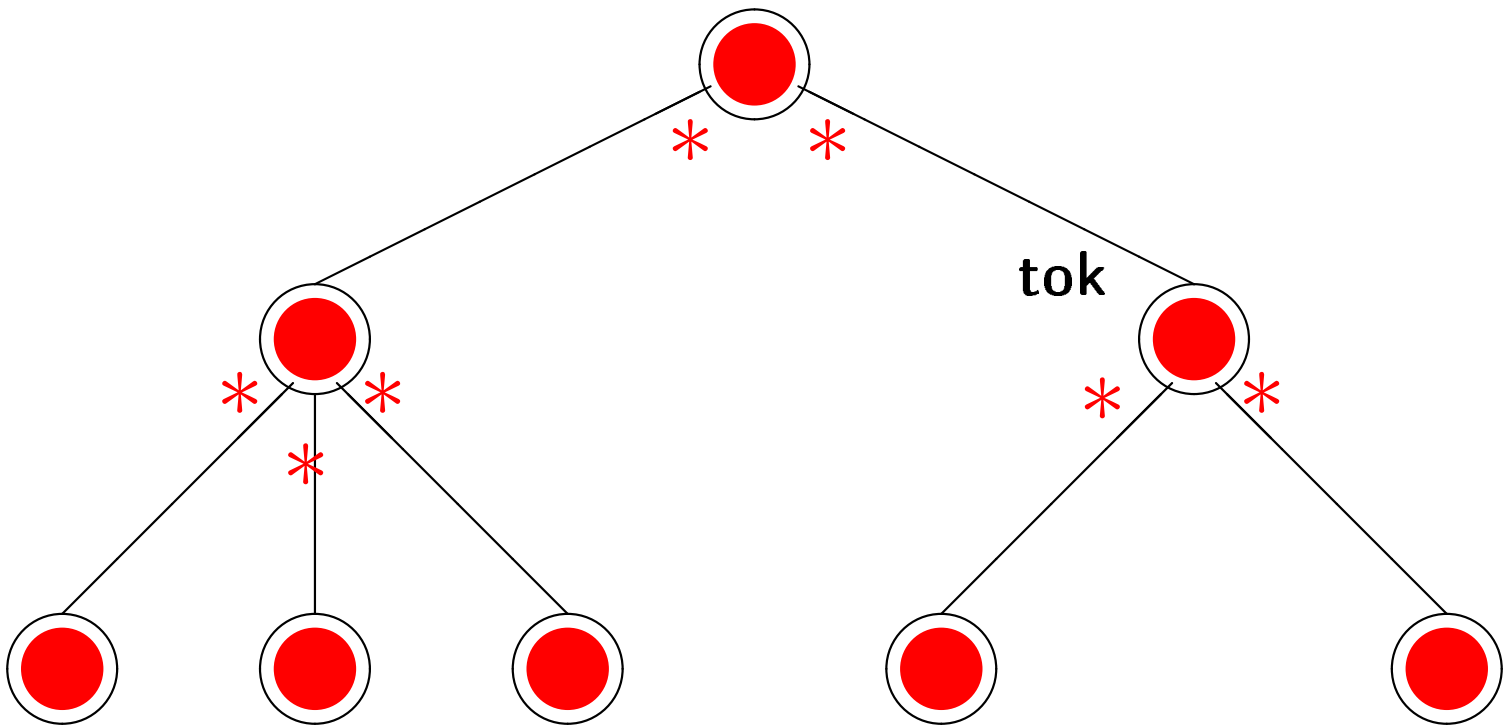
Древесный алгоритм



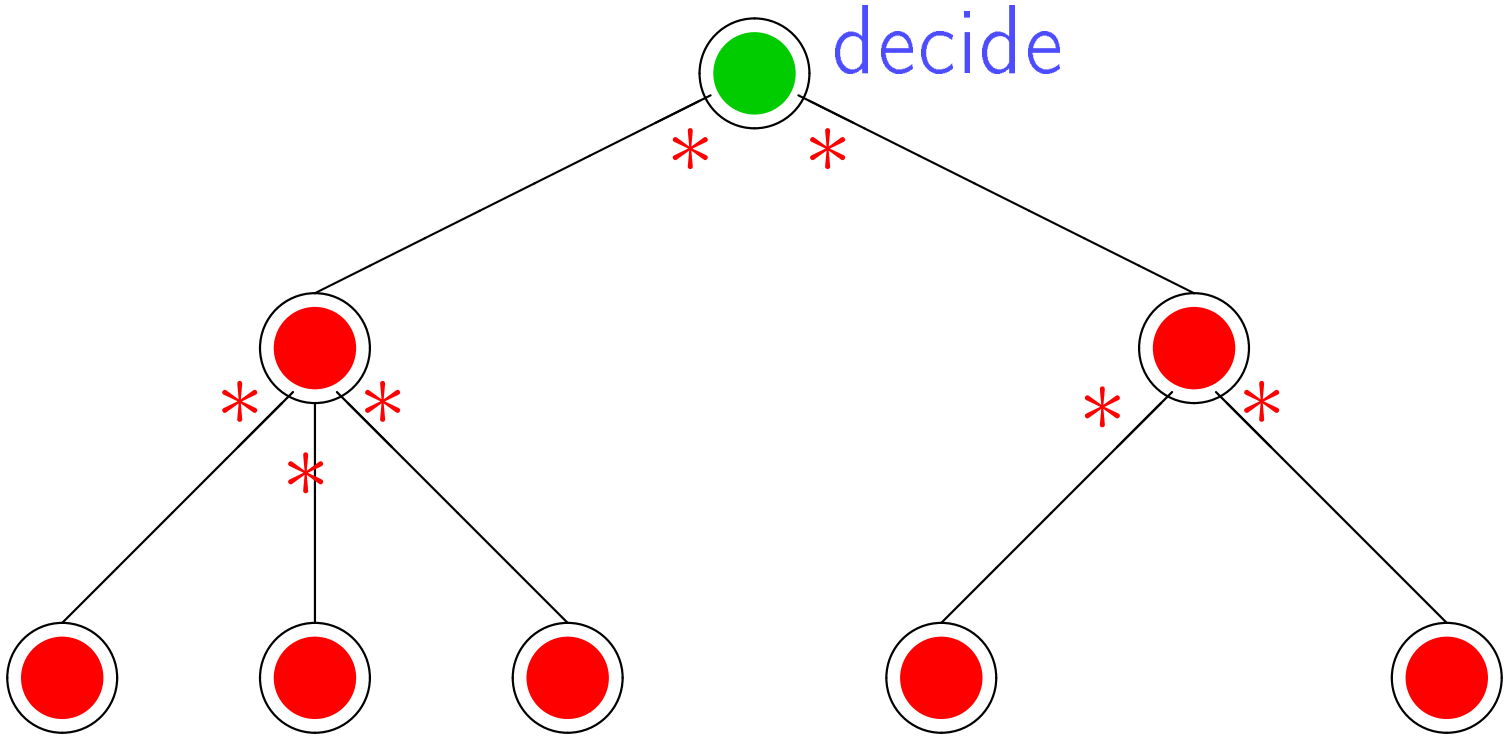
Древесный алгоритм



Древесный алгоритм



Древесный алгоритм



Древесный алгоритм

Будем использовать запись f_{pq} для обозначения события отправления сообщения процессом p процессу q, а g_{pq} — для обозначения события приема процессом q сообщения от процесса p.

Древесный алгоритм

Будем использовать запись f_{pq} для обозначения события отправления сообщения процессом p процессу q, а g_{pq} — для обозначения события приема процессом q сообщения от процесса p.

Обозначим символом T_{pq} множество процессов, которые достижимы из p без прохождения по ребру pq (процессы, лежащие в дереве «по ту же сторону» этого ребра, что и вершина p).

Древесный алгоритм

Будем использовать запись f_{pq} для обозначения события отправления сообщения процессом p процессу q, а g_{pq} — для обозначения события приема процессом q сообщения от процесса p.

Обозначим символом T_{pq} множество процессов, которые достижимы из p без прохождения по ребру pq (процессы, лежащие в дереве «по ту же сторону» этого ребра, что и вершина p).

Из связности сети следуют соотношения

$$T_{pq} = \bigcup_{r \in Neigh_p \setminus \{q\}} T_{rp} \cup \{p\} \quad \text{и} \quad \mathbb{P} = \{p\} \cup \bigcup_{r \in Neigh_p} T_{rp}.$$

Древесный алгоритм

var $rec_p[q]$ for each $q \in Neigh_p$: boolean **init** false ;
(* $rec_p[q]$ принимает значение true,
если p уже получил сообщение от q *)

begin **while** $\#\{q : rec_p[q] =$ false$\} > 1$ **do**
begin receive **tok** from q ; $rec_p[q] :=$ true **end**;
(* Теперь есть *единственный* q_0,
для которого $rec_p[q_0]$ имеет значение false *)
send **tok** to q_0 with $rec_p[q_0] = \mathrm{false}$;
option: receive **tok** from q_0 ; $rec_p[q_0] :=$ true ;
decide
(* Оповестить другие процессы о решении:
forall $q \in Neigh_p, q \neq q_0$ **do** send **tok** to q *)
end

Древесный алгоритм

Утверждение 13.

Древесный алгоритм является волновым алгоритмом.

Древесный алгоритм

Утверждение 13.

Древесный алгоритм является волновым алгоритмом.

Доказательство.

1) Каждый процесс отправляет не более одного сообщения.
Значит, в алгоритме используется не более N сообщений.
Отсюда следует, что алгоритм достигает заключительной конфигурации γ спустя конечное число шагов.
Покажем, что в γ хотя бы в одном из процессов уже произошло событие **decide**

Древесный алгоритм

Доказательство.

2) Суммарное число битов во всех массивах *rec* равно удвоенному числу каналов связи, т.е. равно $2(N-1)$.

Древесный алгоритм

Доказательство.

2) Суммарное число битов во всех массивах *rec* равно удвоенному числу каналов связи, т.е. равно 2(N-1) .

Пусть

F — это число битов *rec* со значением false в конфигурации γ ,

K — количество процессов, которые уже отправили сообщения в конфигурации γ . Прием каждого отправленного сообщения обращает один из битов массива *rec* в true .

Древесный алгоритм

Доказательство.

2) Суммарное число битов во всех массивах *rec* равно удвоенному числу каналов связи, т.е. равно 2(N-1) .

Пусть

F — это число битов *rec* со значением false в конфигурации γ ,

K — количество процессов, которые уже отправили сообщения в конфигурации γ . Прием каждого отправленного сообщения обращает один из битов массива *rec* в true .

Таким образом, общее число битов *rec* равно $2N - 2$, и K из них могут иметь значение true . В заключительной конфигурации γ никаких сообщений не пересылается, и поэтому $F = (2N - 2) - K$.

Древесный алгоритм

Доказательство.

3) Допустим, что ни один из процессов не принял решения в γ.

Древесный алгоритм

Доказательство.

3) Допустим, что ни один из процессов не принял решения в γ.

Тогда $N - K$ процессов, которые в γ еще не отправляли сообщения, имеют в *rec* по меньшей мере два бита со значением false Почему?

Древесный алгоритм

Доказательство.

3) Допустим, что ни один из процессов не принял решения в γ.

Тогда $N - K$ процессов, которые в γ еще не отправляли сообщения, имеют в *rec* по меньшей мере два бита со значением false Иначе они могли бы отправить сообщение, хотя γ – заключительная конфигурация.

Древесный алгоритм

Доказательство.

3) Допустим, что ни один из процессов не принял решения в γ.

Тогда $N - K$ процессов, которые в γ еще не отправляли сообщения, имеют в *rec* по меньшей мере два бита со значением false Иначе они могли бы отправить сообщение, хотя γ – заключительная конфигурация.
Кроме того, K процессов, которые в γ уже отправили сообщение, имеют в *rec* хотя бы по одному биту со значением false . Почему?

Древесный алгоритм

Доказательство.

3) Допустим, что ни один из процессов не принял решения в γ.

Тогда $N - K$ процессов, которые в γ еще не отправляли сообщения, имеют в *rec* по меньшей мере два бита со значением false Иначе они могли бы отправить сообщение, хотя γ – заключительная конфигурация.
Кроме того, K процессов, которые в γ уже отправили сообщение, имеют в *rec* хотя бы по одному биту со значением false. Иначе они могли бы принять решение вопреки сделанному предположению о γ.

Древесный алгоритм

Доказательство.

3) Допустим, что ни один из процессов не принял решения в γ.

Тогда $N - K$ процессов, которые в γ еще не отправляли сообщения, имеют в *rec* по меньшей мере два бита со значением false Иначе они могли бы отправить сообщение, хотя γ – заключительная конфигурация.
Кроме того, K процессов, которые в γ уже отправили сообщение, имеют в *rec* хотя бы по одному биту со значением false. Иначе они могли бы принять решение вопреки сделанному предположению о γ.
Значит, $F \geq 2(N - K) + K$.

Древесный алгоритм

Доказательство.

3) Допустим, что ни один из процессов не принял решения в γ.

Тогда $N - K$ процессов, которые в γ еще не отправляли сообщения, имеют в *rec* по меньшей мере два бита со значением false Иначе они могли бы отправить сообщение, хотя γ – заключительная конфигурация.
Кроме того, K процессов, которые в γ уже отправили сообщение, имеют в *rec* хотя бы по одному биту со значением false. Иначе они могли бы принять решение вопреки сделанному предположению о γ.
Значит, $F \geq 2(N - K) + K$.

Ранее мы показали, что $(2N - 2) - K \geq 2(N - K) + K$. Отсюда следует $-2 \geq 0$.

Древесный алгоритм

Доказательство.

3) Допустим, что ни один из процессов не принял решения в γ.

Тогда $N - K$ процессов, которые в γ еще не отправляли сообщения, имеют в *rec* по меньшей мере два бита со значением false Иначе они могли бы отправить сообщение, хотя γ – заключительная конфигурация.
Кроме того, K процессов, которые в γ уже отправили сообщение, имеют в *rec* хотя бы по одному биту со значением false. Иначе они могли бы принять решение вопреки сделанному предположению о γ.
Значит, $F \geq 2(N - K) + K$.

Ранее мы показали, что $(2N - 2) - K \geq 2(N - K) + K$. Отсюда следует $-2 \geq 0$.
Полученное противоречие означает, что хотя бы одно решение в γ уже принято.

Древесный алгоритм

Доказательство.

4) Покажем, что решение предшествует какому-либо событию в каждом из процессов.

Древесный алгоритм

Доказательство.

4) Покажем, что решение предшествует какому-либо событию в каждом из процессов.

Покажем индукцией по числу событий приема, что

$$\forall\, s \in T_{pq}\ \exists e \in C_s : e \preceq g_{pq}.$$

Древесный алгоритм

Доказательство.

4) Покажем, что решение предшествует какому-либо событию в каждом из процессов.

Покажем индукцией по числу событий приема, что

$$\forall\, s \in T_{pq}\ \exists e \in C_s : e \preceq g_{pq}.$$

Допустим, это верно для всех событий приема, предшествующих событию g_{pq}. Тогда событию g_{pq} предшествует событие f_{pq} (в процессе p). Поэтому из устройства программы p следует, что для всех $r \in Neigh_p$ при $r \neq q$ событию f_{pq} предшествует событие g_{rp}.

Древесный алгоритм

Доказательство.

4) Покажем, что решение предшествует какому-либо событию в каждом из процессов.

Покажем индукцией по числу событий приема, что

$$\forall\, s \in T_{pq}\ \exists e \in C_s : e \preceq g_{pq}.$$

Допустим, это верно для всех событий приема, предшествующих событию g_{pq}. Тогда событию g_{pq} предшествует событие f_{pq} (в процессе p). Поэтому из устройства программы p следует, что для всех $r \in Neigh_p$ при $r \neq q$ событию f_{pq} предшествует событие g_{rp}.
Из индуктивной гипотезы вытекает, что для всех таких r и для всех $s \in T_{rp}$ найдется такое событие $e \in C_s$, что $e \preceq g_{rp}$.
Значит, $e \preceq g_{pq}$.

Древесный алгоритм

Доказательство.

5) Решению $decide_p$ в узле p предшествует событие g_{rp} для всех $r \in Neigh_p$.

Отсюда следует, что

$$\forall\, s \in \mathbb{P}\ \exists e \in C_s : e \preceq d_p.$$

□

Алгоритм эха

Алгоритм эха — это централизованный волновой алгоритм для сетей с произвольной топологией.

Алгоритм наводняет сообщениями **tok** все процессы, выделяя таким образом остовное дерево. Маркеры возвращаются «эхом» обратно по ребрам этого дерева, напоминая поток сообщений в древесном алгоритме. Сценарий работы таков.

Алгоритм эха

Алгоритм эха — это централизованный волновой алгоритм для сетей с произвольной топологией.

Алгоритм наводняет сообщениями **tok** все процессы, выделяя таким образом остовное дерево. Маркеры возвращаются «эхом» обратно по ребрам этого дерева, напоминая поток сообщений в древесном алгоритме. Сценарий работы таков.

1) Инициатор отправляет сообщение всем своим соседям.

Алгоритм эха

Алгоритм эха — это централизованный волновой алгоритм для сетей с произвольной топологией.

Алгоритм наводняет сообщениями **tok** все процессы, выделяя таким образом остовное дерево. Маркеры возвращаются «эхом» обратно по ребрам этого дерева, напоминая поток сообщений в древесном алгоритме. Сценарий работы таков.

1) Инициатор отправляет сообщение всем своим соседям.

2) После получения первого сообщения всякий неинициатор переправляет сообщения всем своим соседям, за исключением того, от которого было получено это сообщение.

Алгоритм эха

Алгоритм эха — это централизованный волновой алгоритм для сетей с произвольной топологией.

Алгоритм наводняет сообщениями **tok** все процессы, выделяя таким образом остовное дерево. Маркеры возвращаются «эхом» обратно по ребрам этого дерева, напоминая поток сообщений в древесном алгоритме. Сценарий работы таков.

1) Инициатор отправляет сообщение всем своим соседям.

2) После получения первого сообщения всякий неинициатор переправляет сообщения всем своим соседям, за исключением того, от которого было получено это сообщение.

3) Как только неинициатор получит сообщения от всех своих соседей, он отправляет эхо родительскому процессу.

Алгоритм эха

Алгоритм эха — это централизованный волновой алгоритм для сетей с произвольной топологией.

Алгоритм наводняет сообщениями **tok** все процессы, выделяя таким образом остовное дерево. Маркеры возвращаются «эхом» обратно по ребрам этого дерева, напоминая поток сообщений в древесном алгоритме. Сценарий работы таков.

1) Инициатор отправляет сообщение всем своим соседям.

2) После получения первого сообщения всякий неинициатор переправляет сообщения всем своим соседям, за исключением того, от которого было получено это сообщение.

3) Как только неинициатор получит сообщения от всех своих соседей, он отправляет эхо родительскому процессу.

4) Как только инициатор получит сообщения от всех своих соседей, он принимает решение.

Алгоритм эха

var rec_p : integer **init** 0; (*Подсчет числа принятых сообщений*)
$father_p$: $\mathbb{P}$ **init** *undef* ;

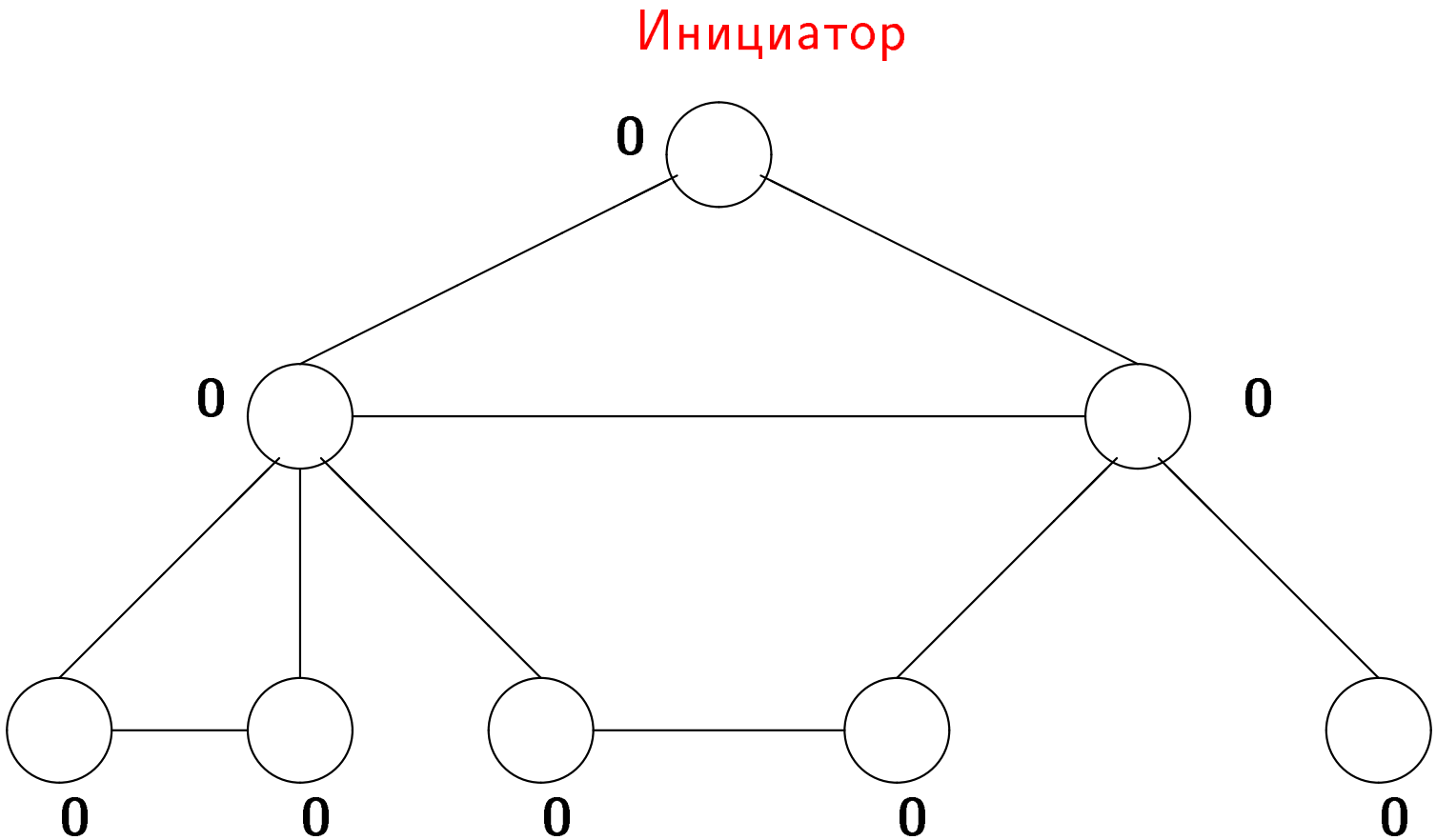
Для инициатора:

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

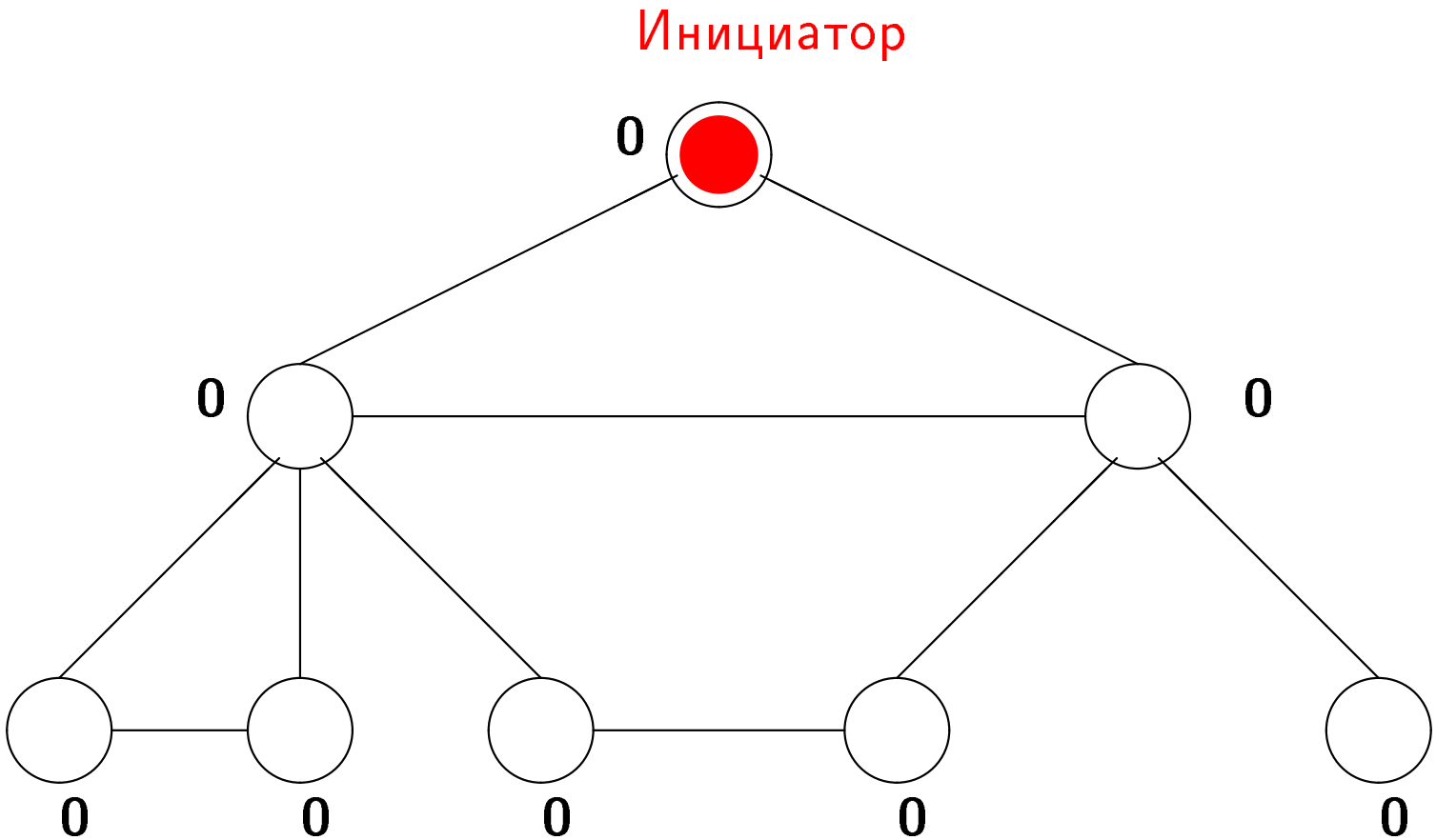
Для неинициатора:

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

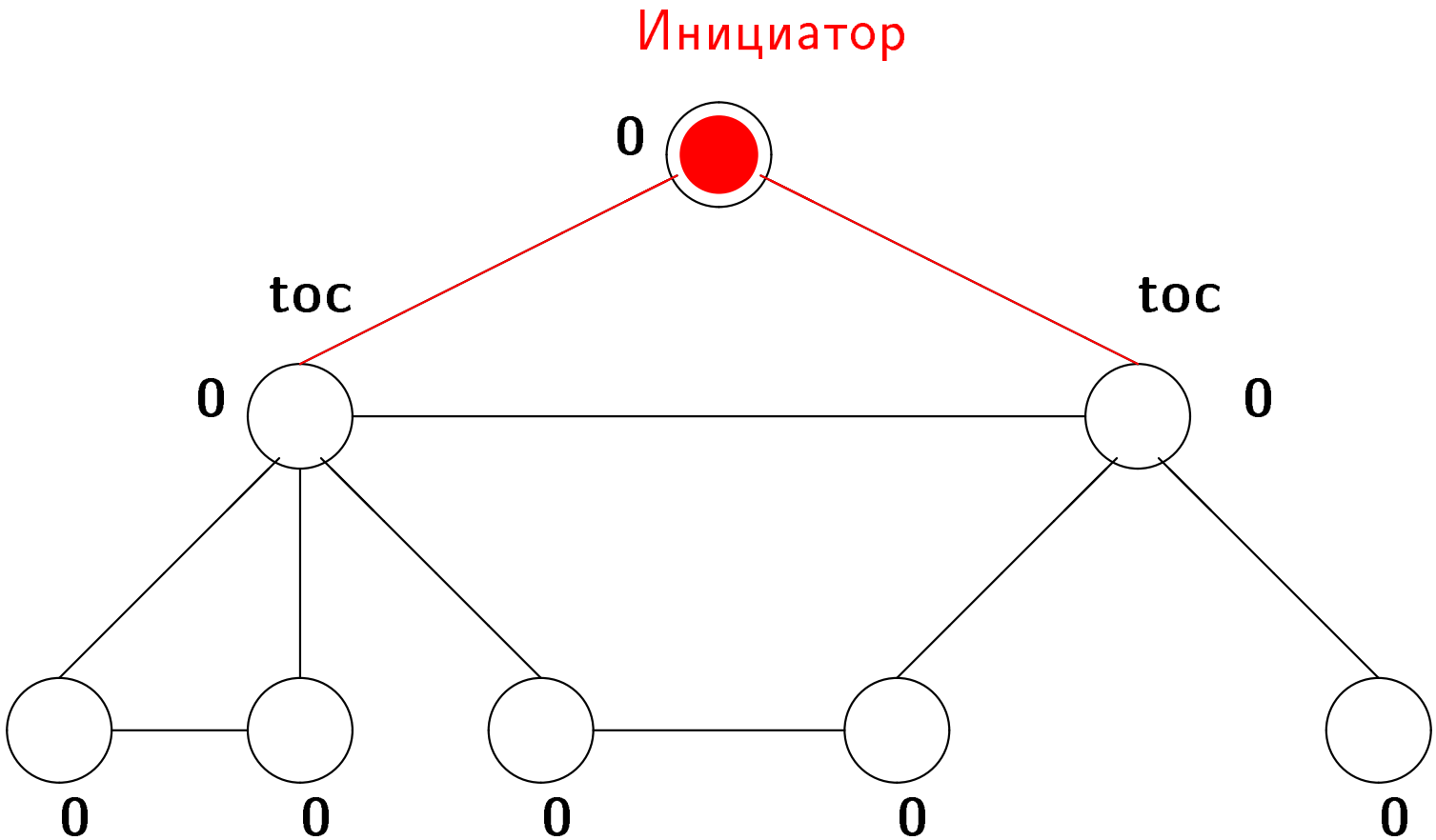
Алгоритм эха



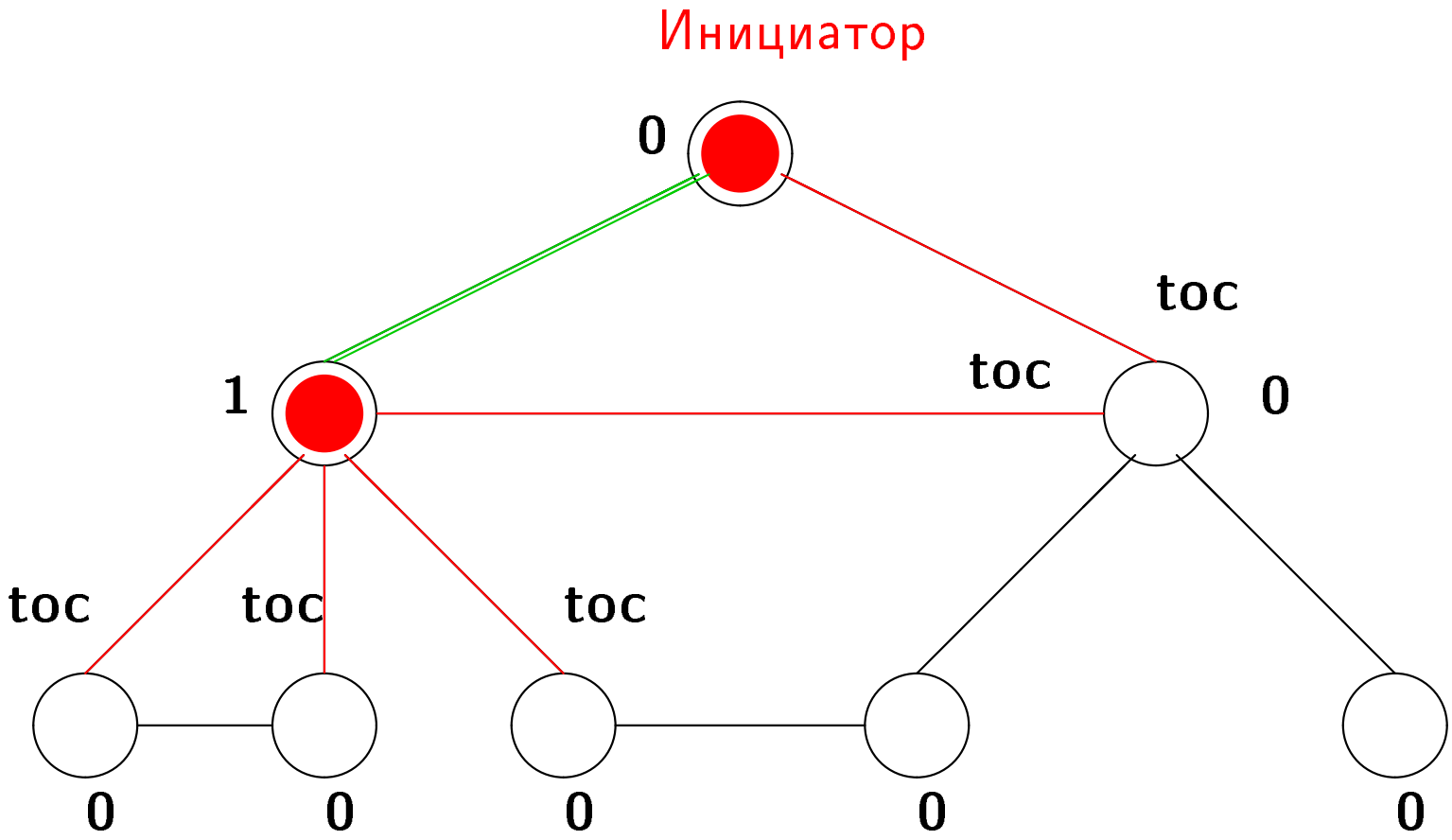
Алгоритм эха



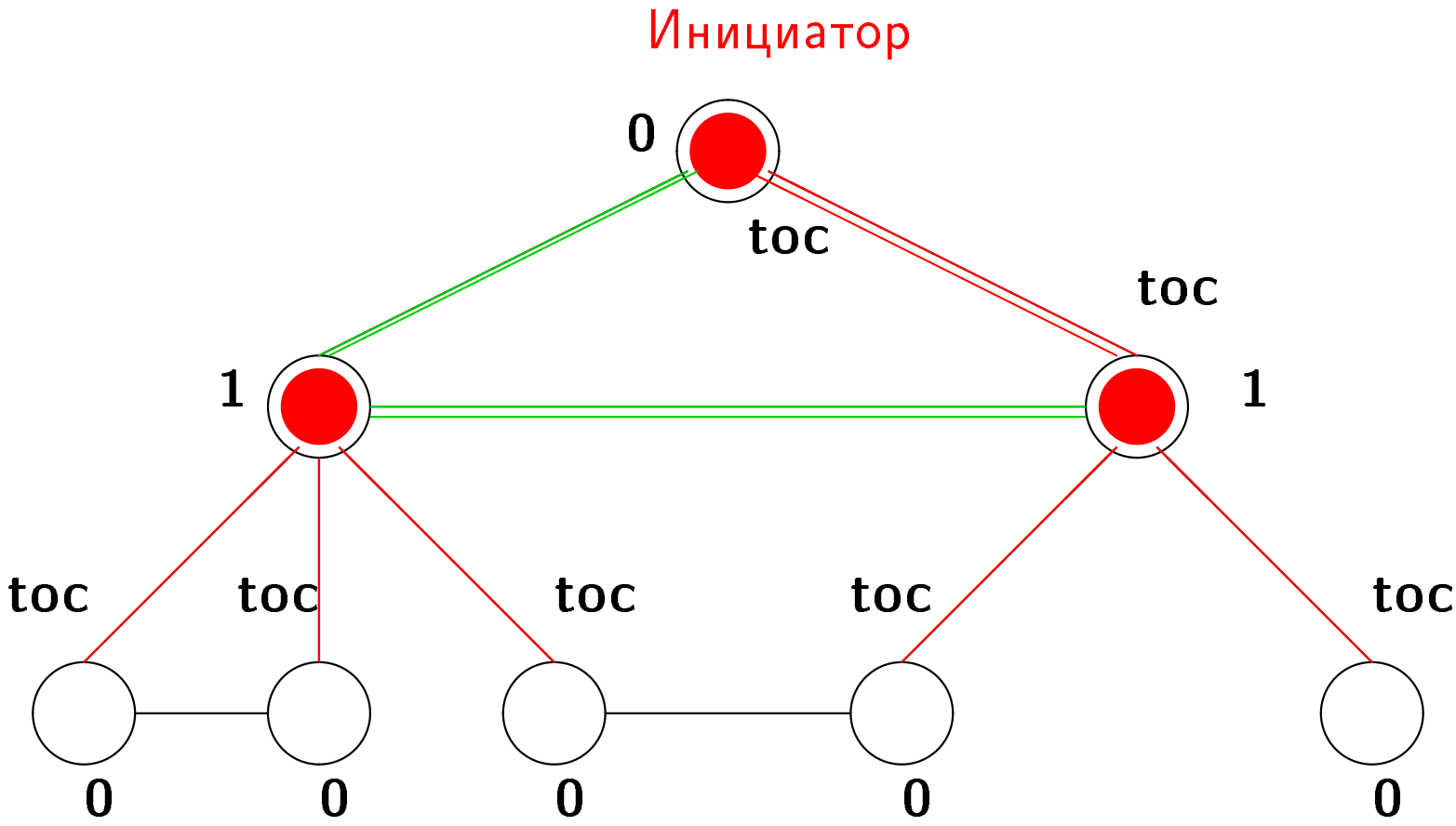
Алгоритм эха



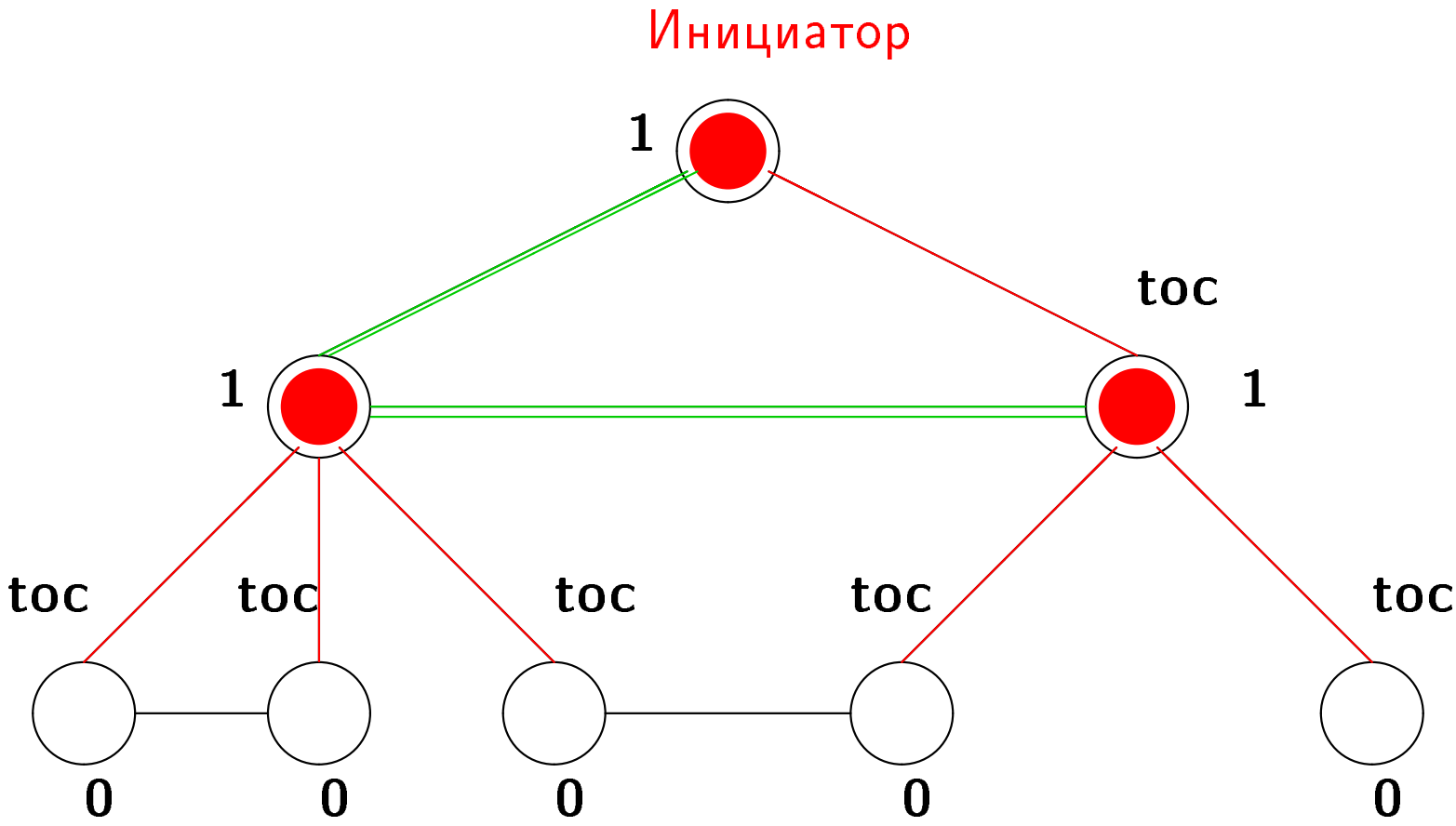
Алгоритм эха



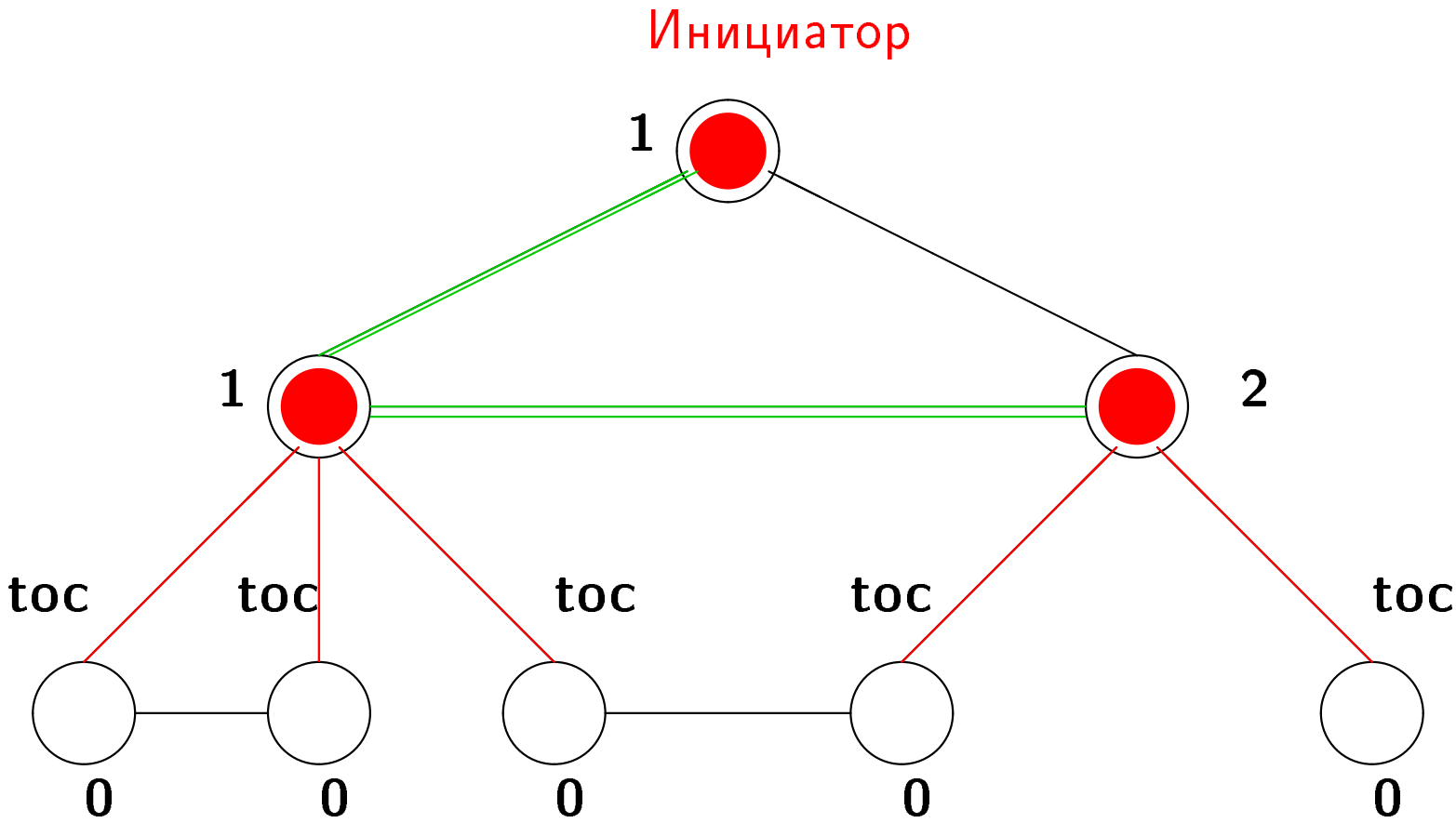
Алгоритм эха



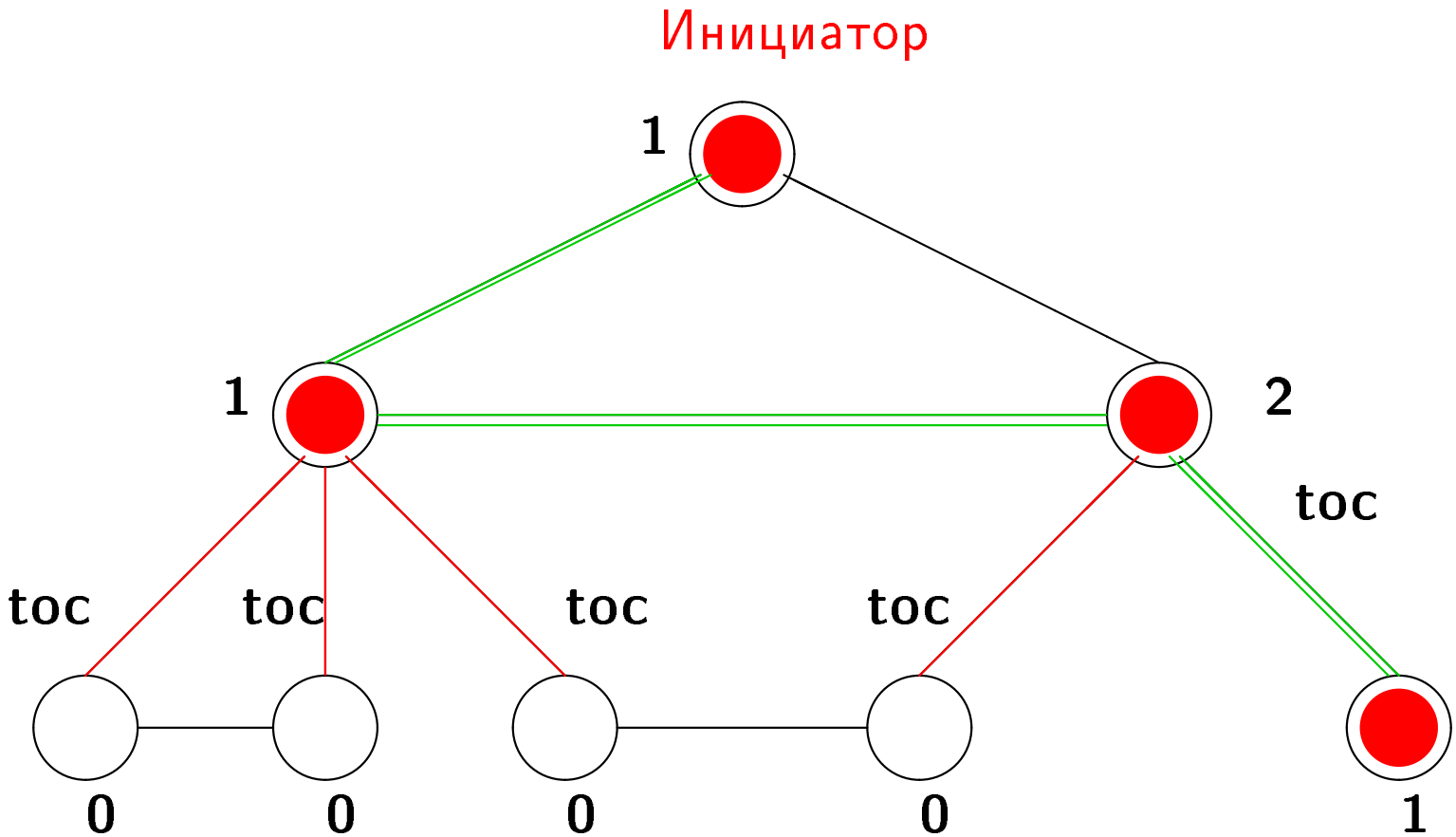
Алгоритм эха



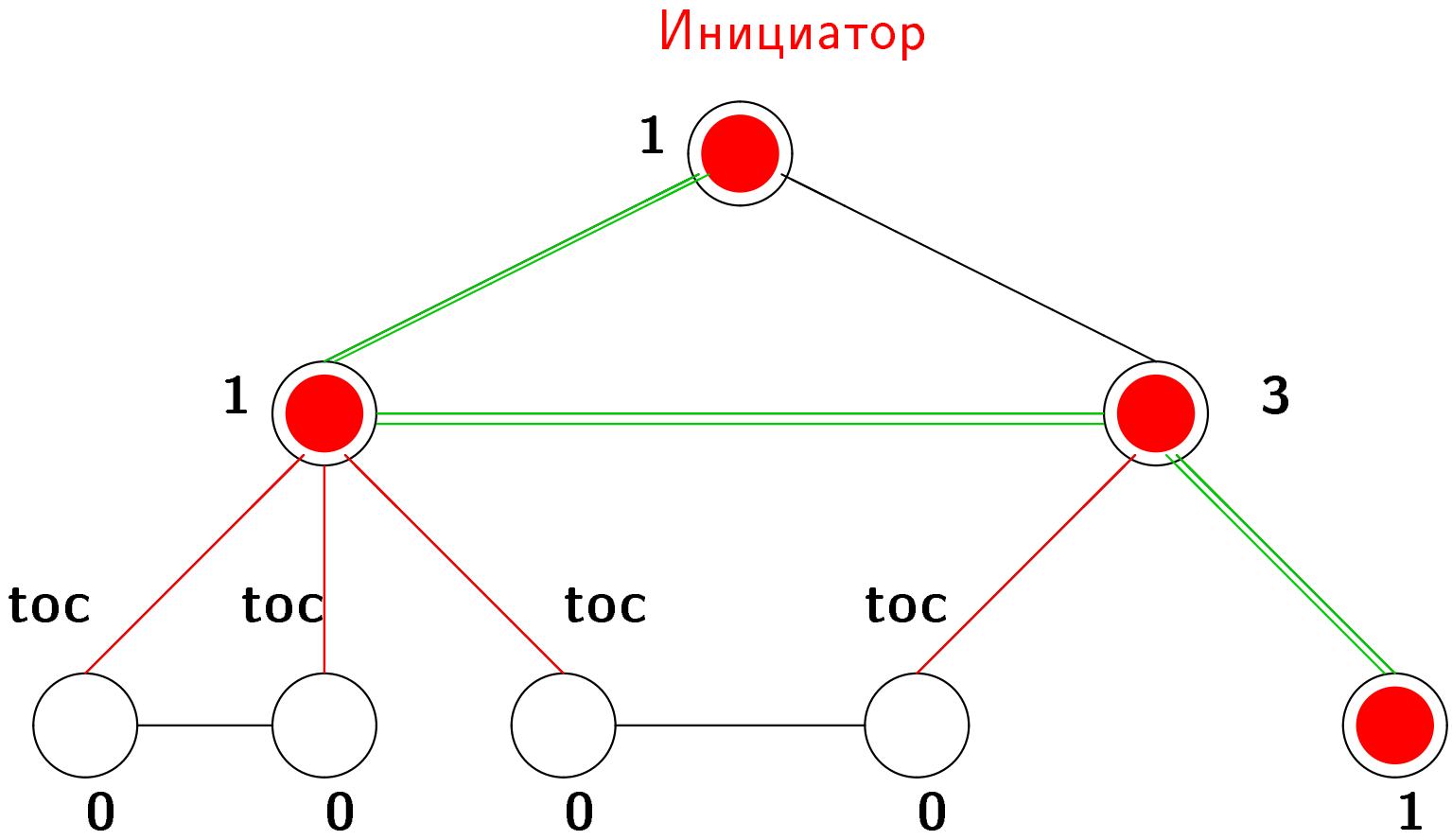
Алгоритм эха



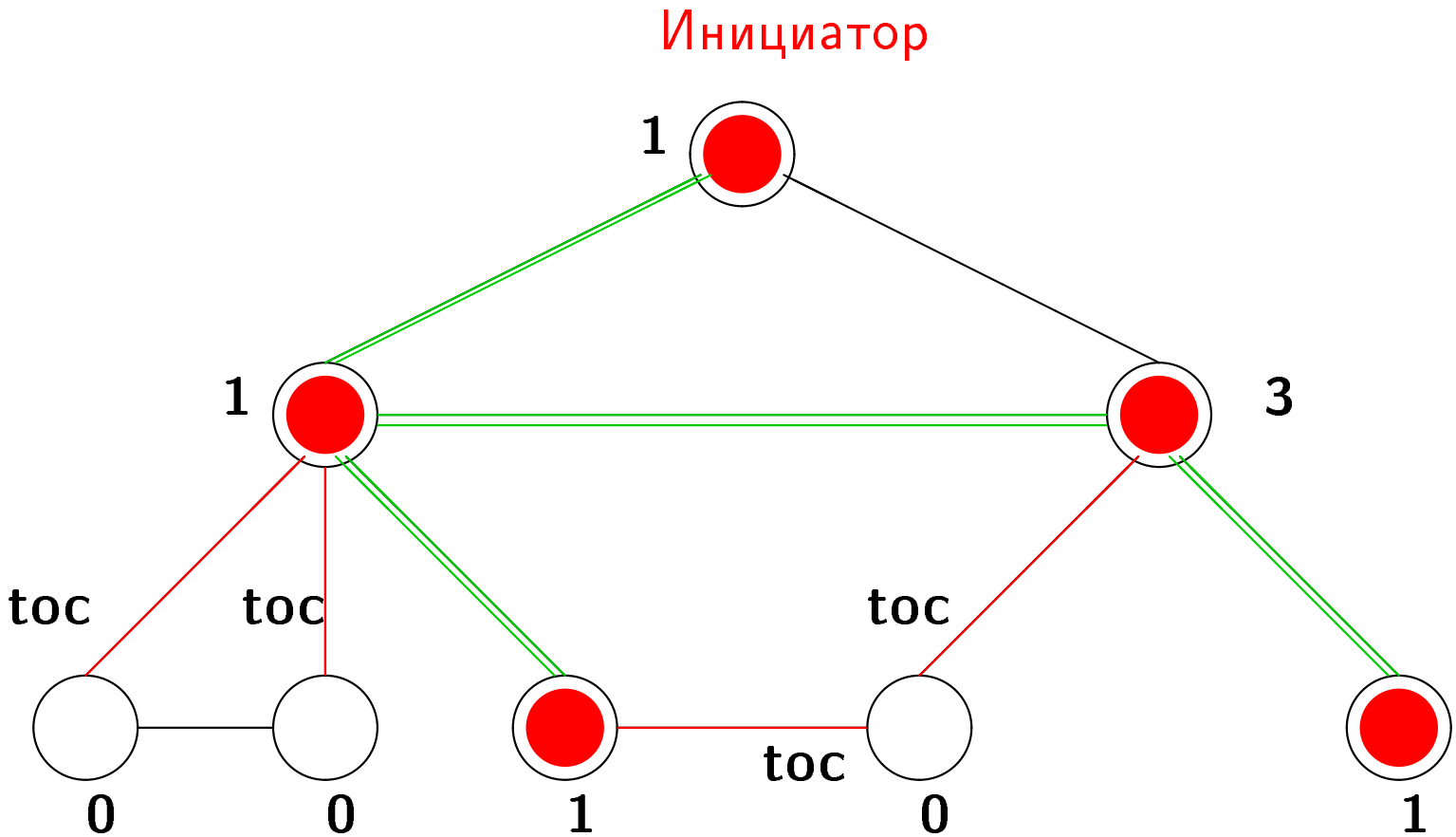
Алгоритм эха



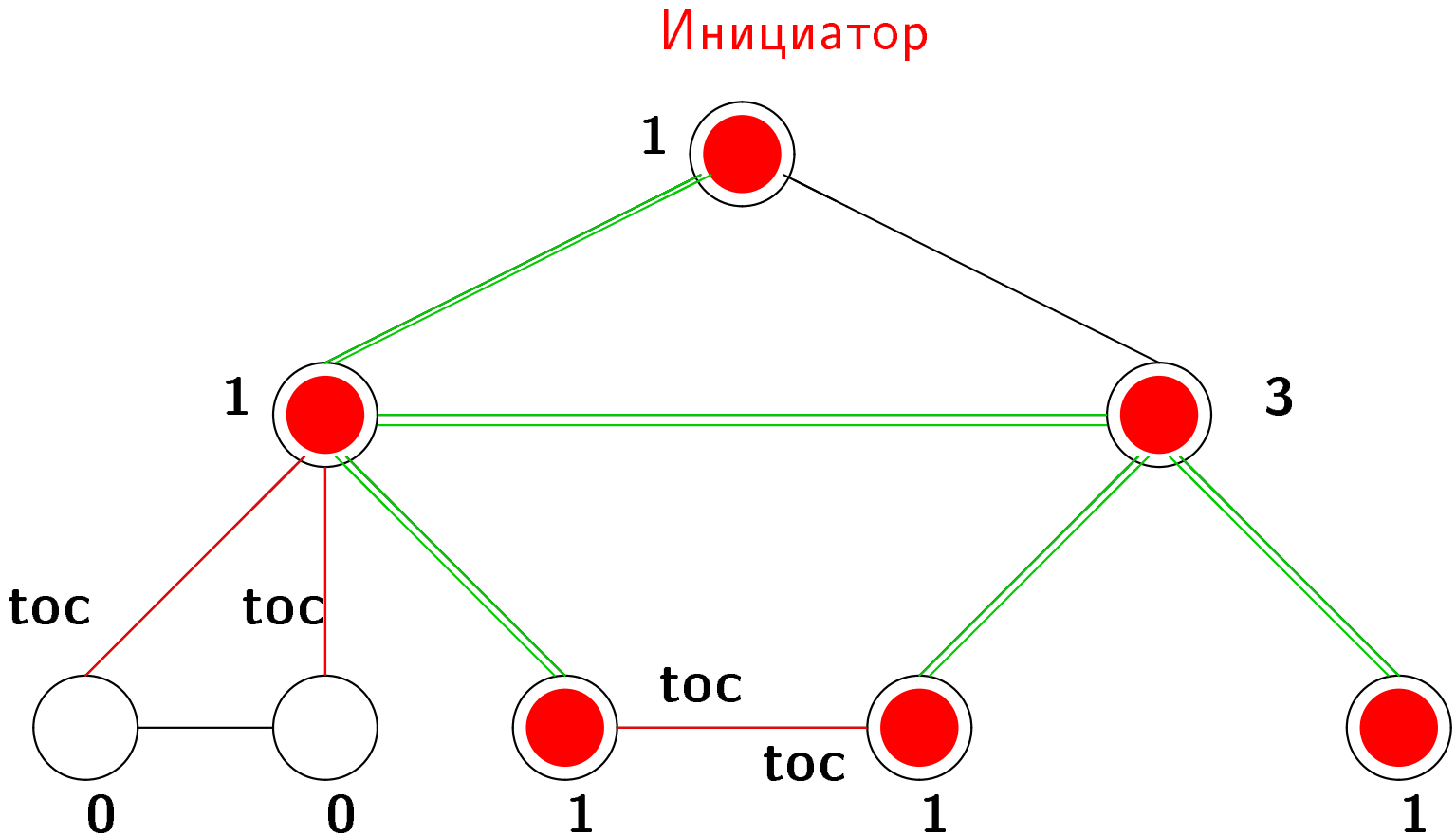
Алгоритм эха



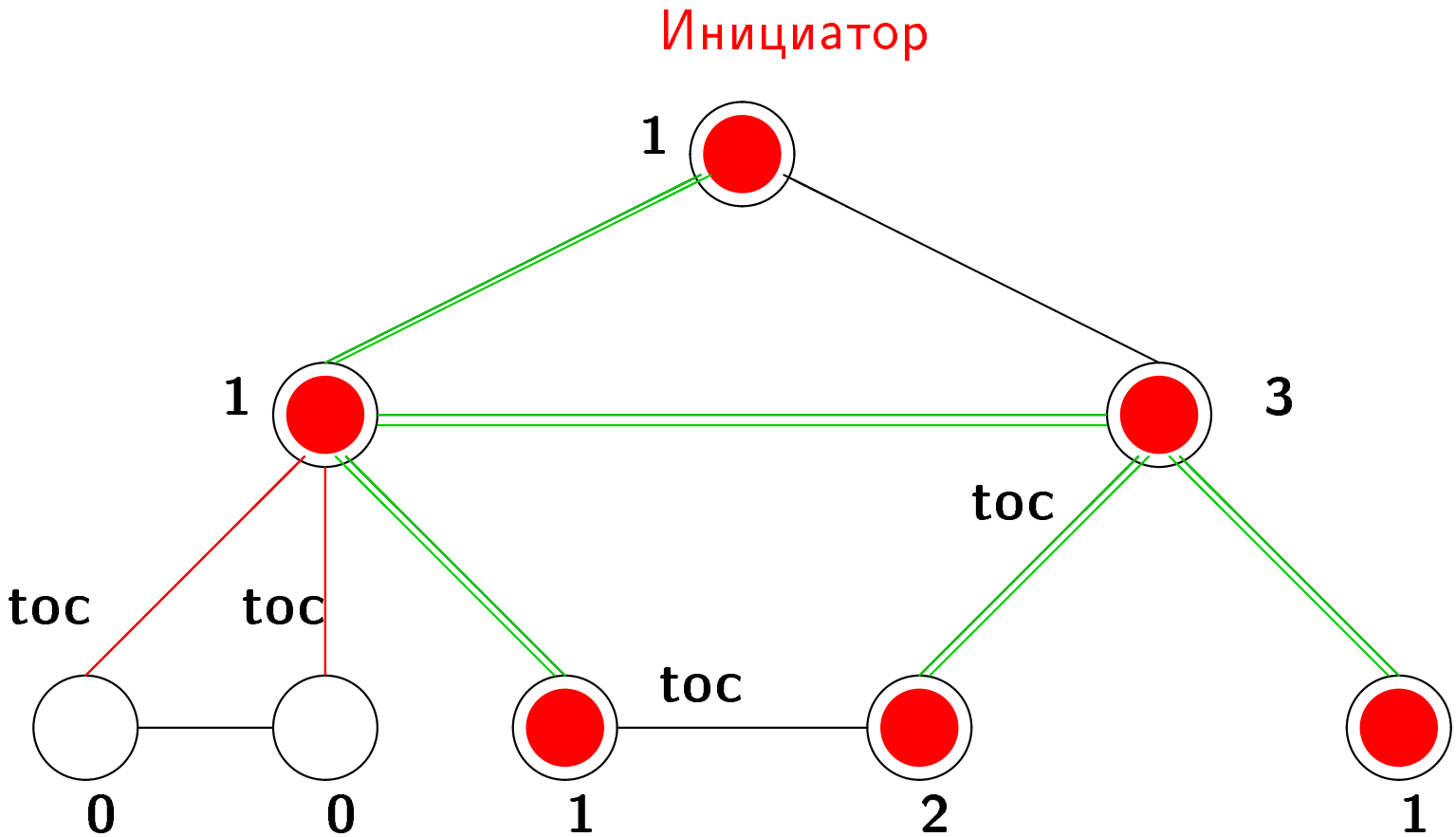
Алгоритм эха



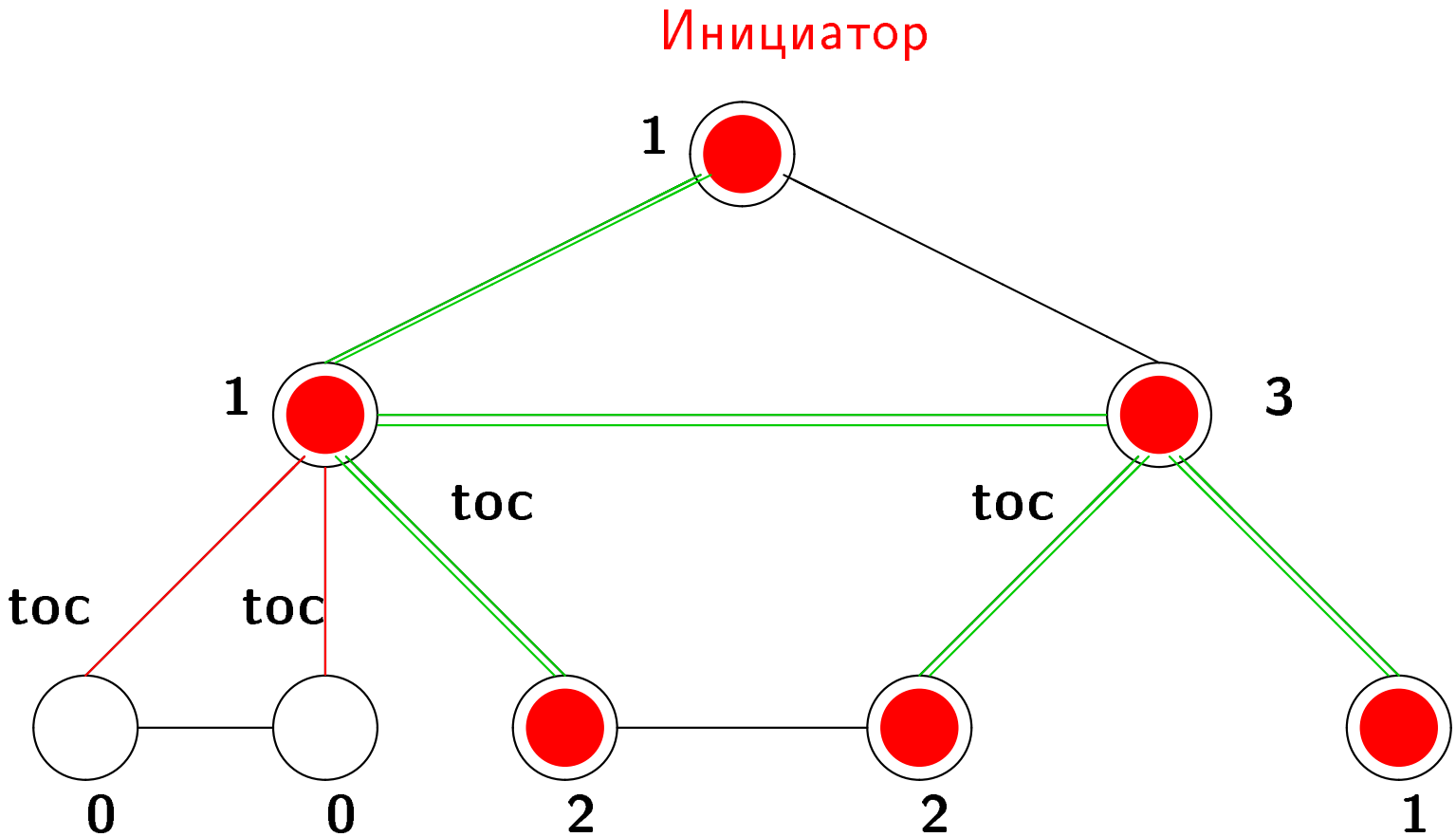
Алгоритм эха



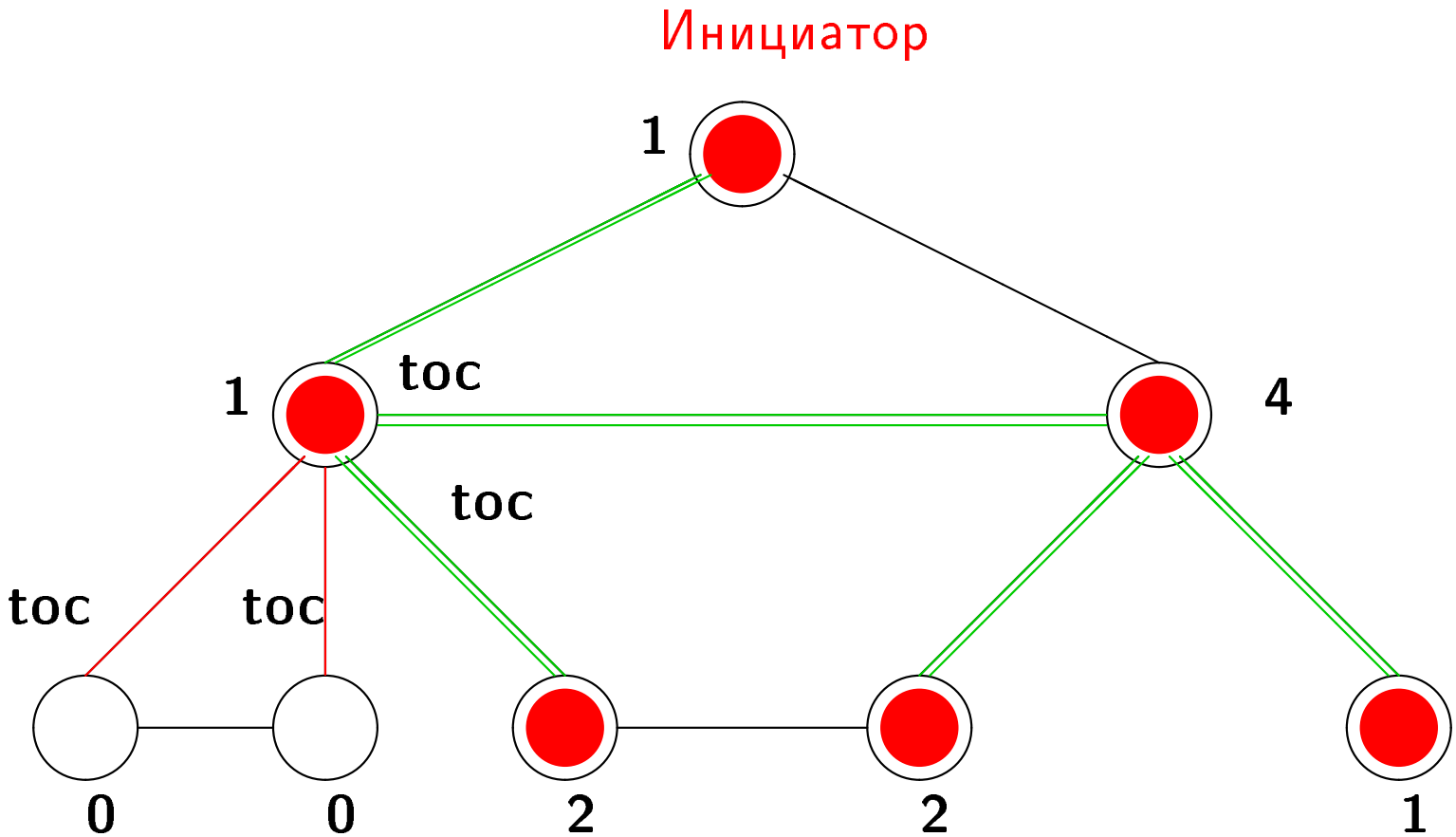
Алгоритм эха



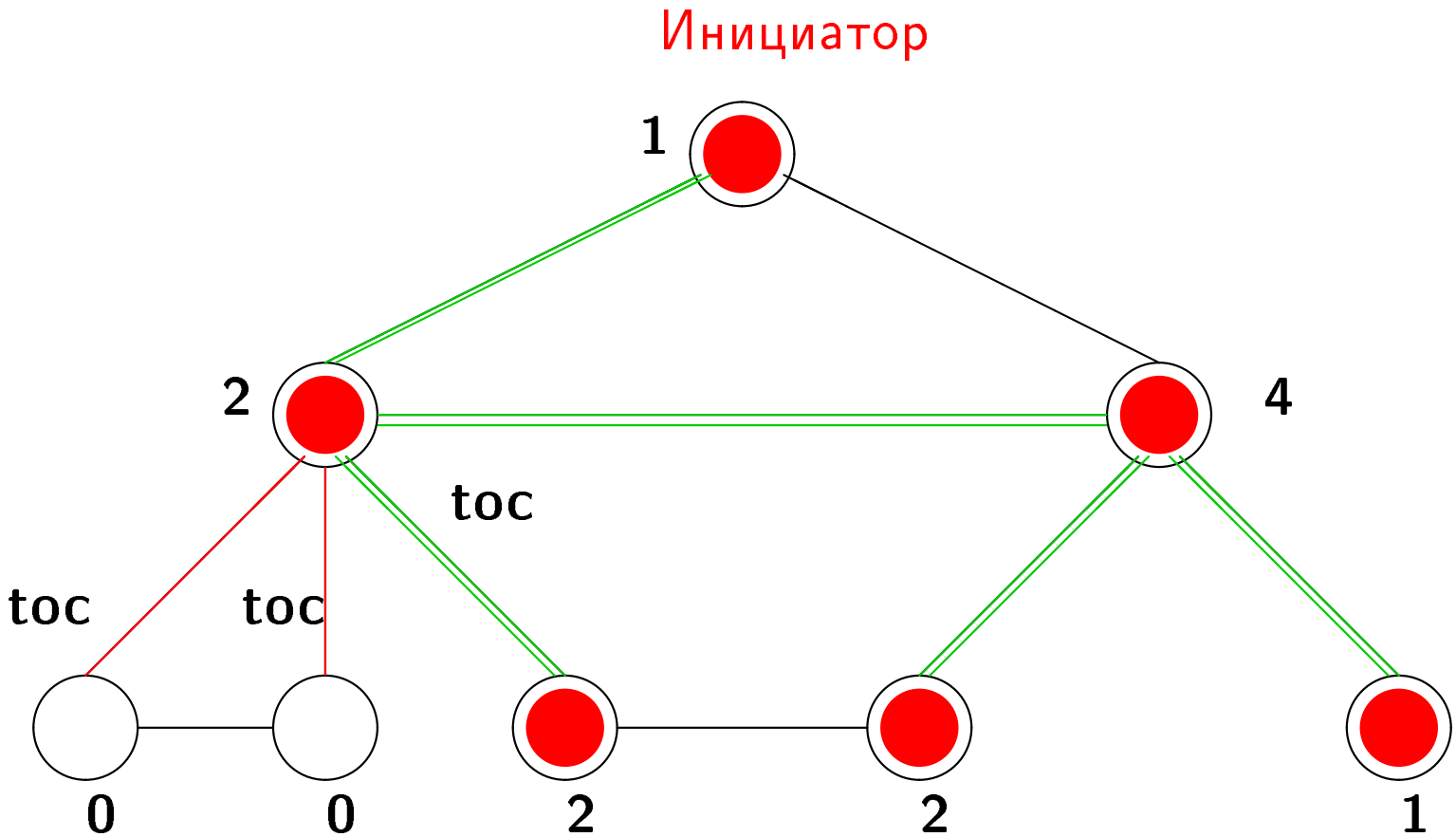
Алгоритм эха



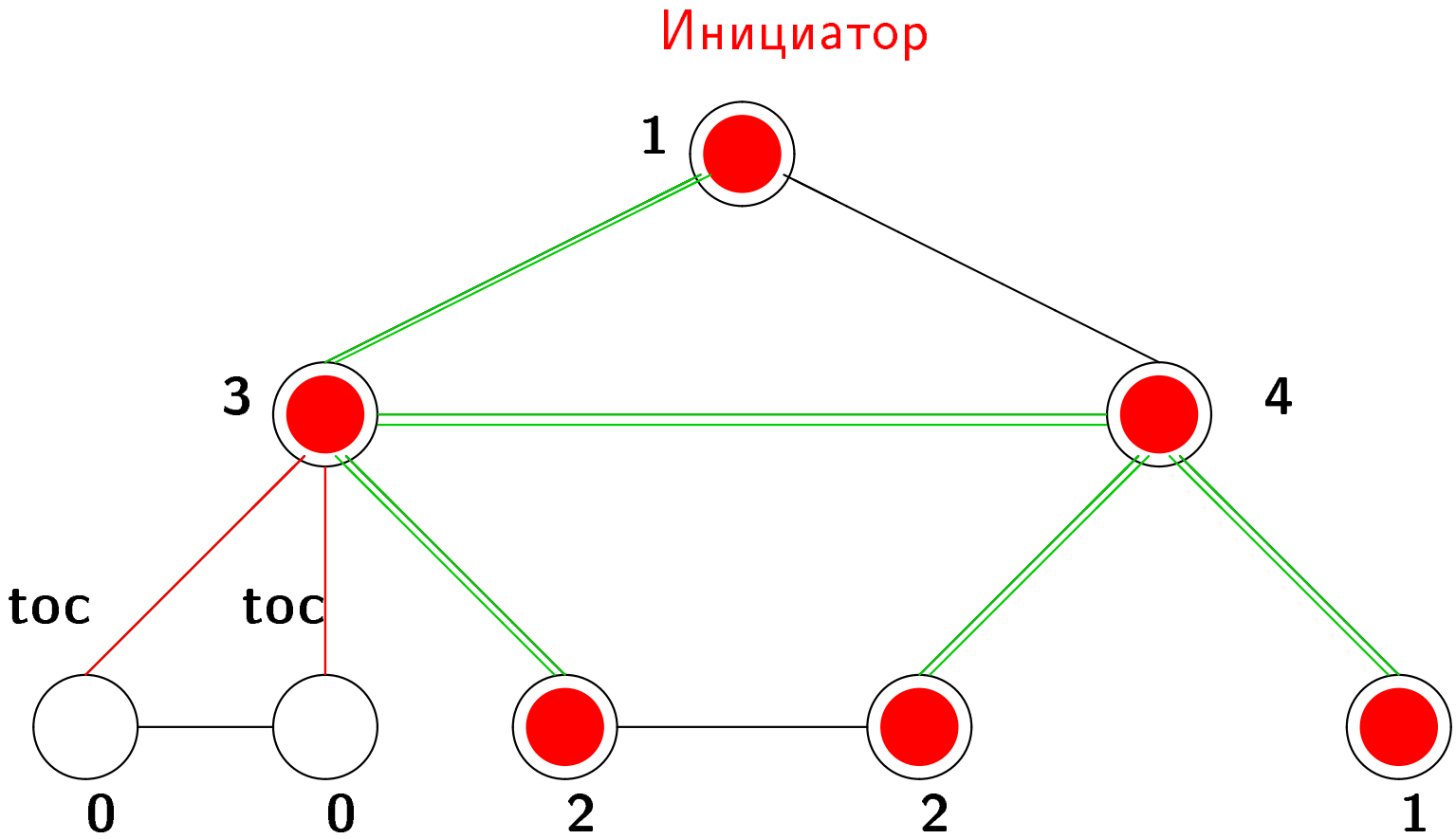
Алгоритм эха



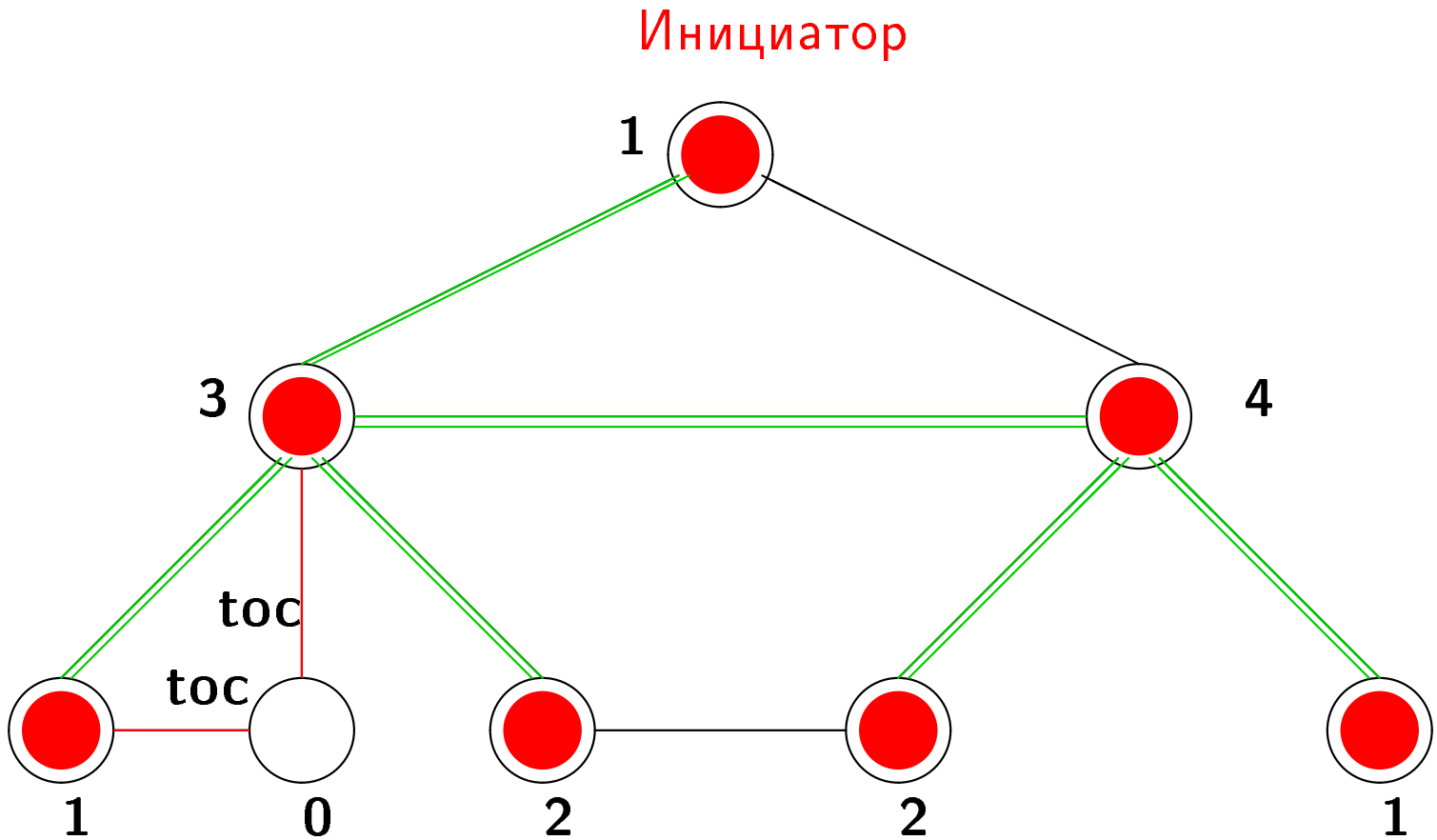
Алгоритм эха



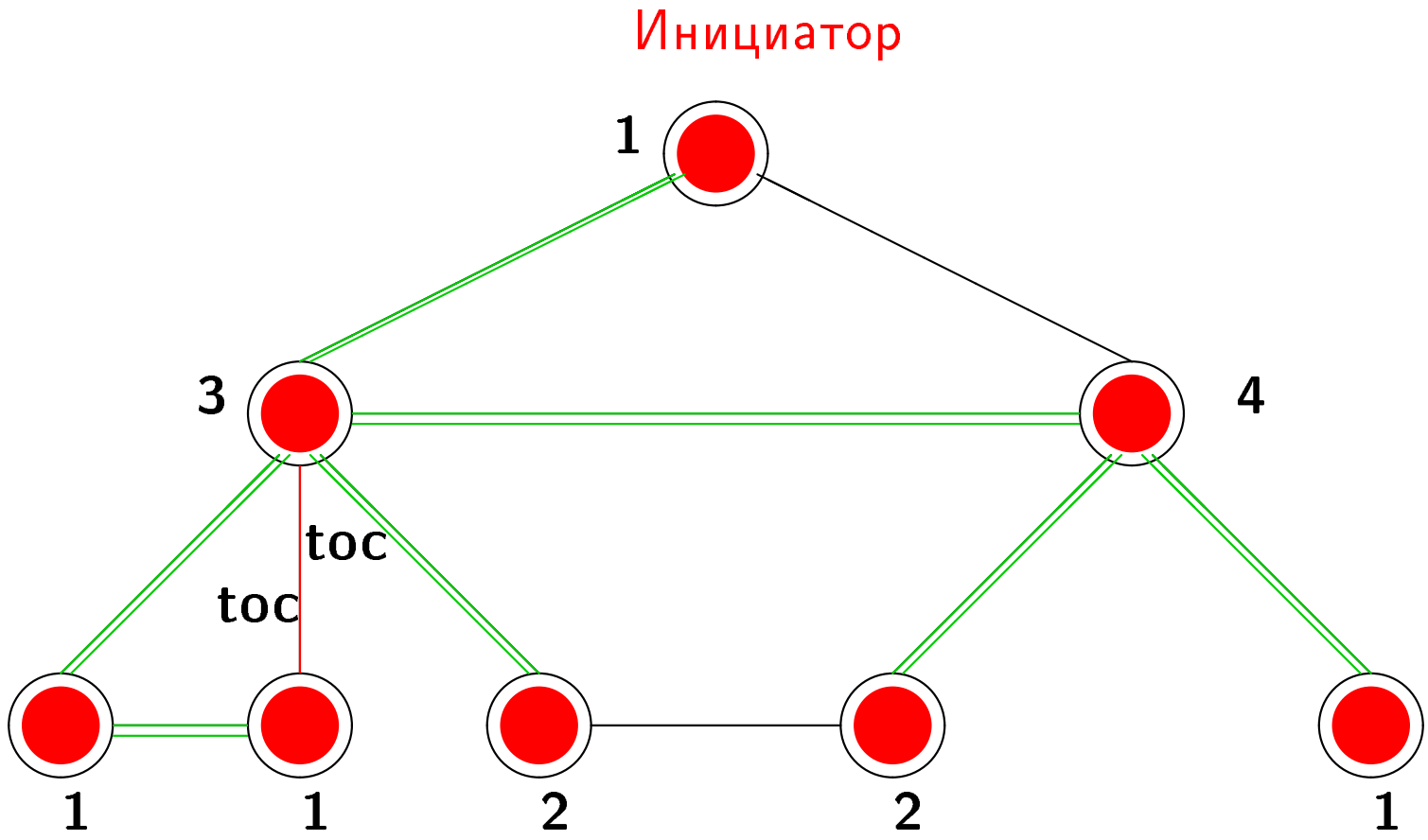
Алгоритм эха



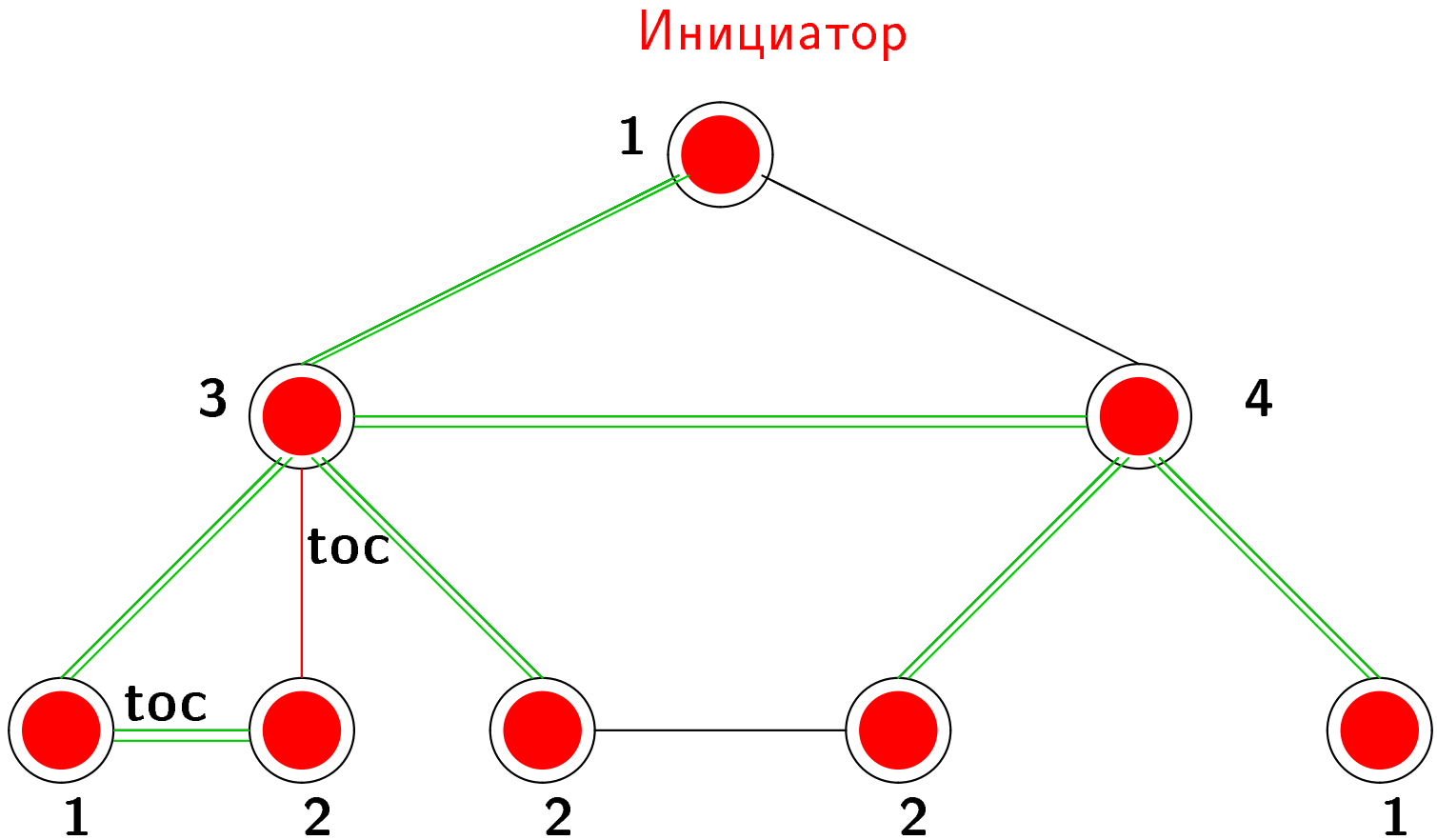
Алгоритм эха



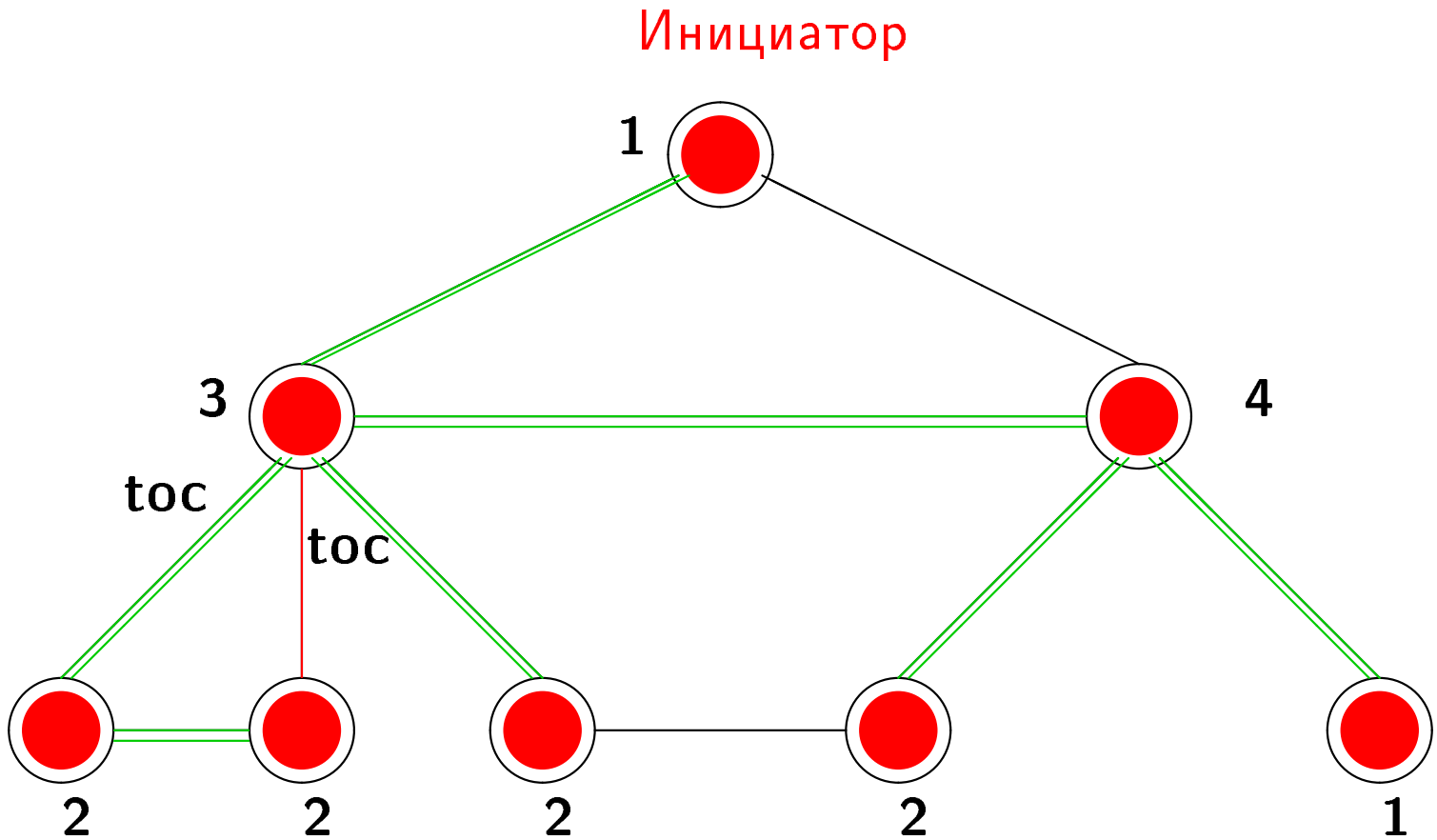
Алгоритм эха



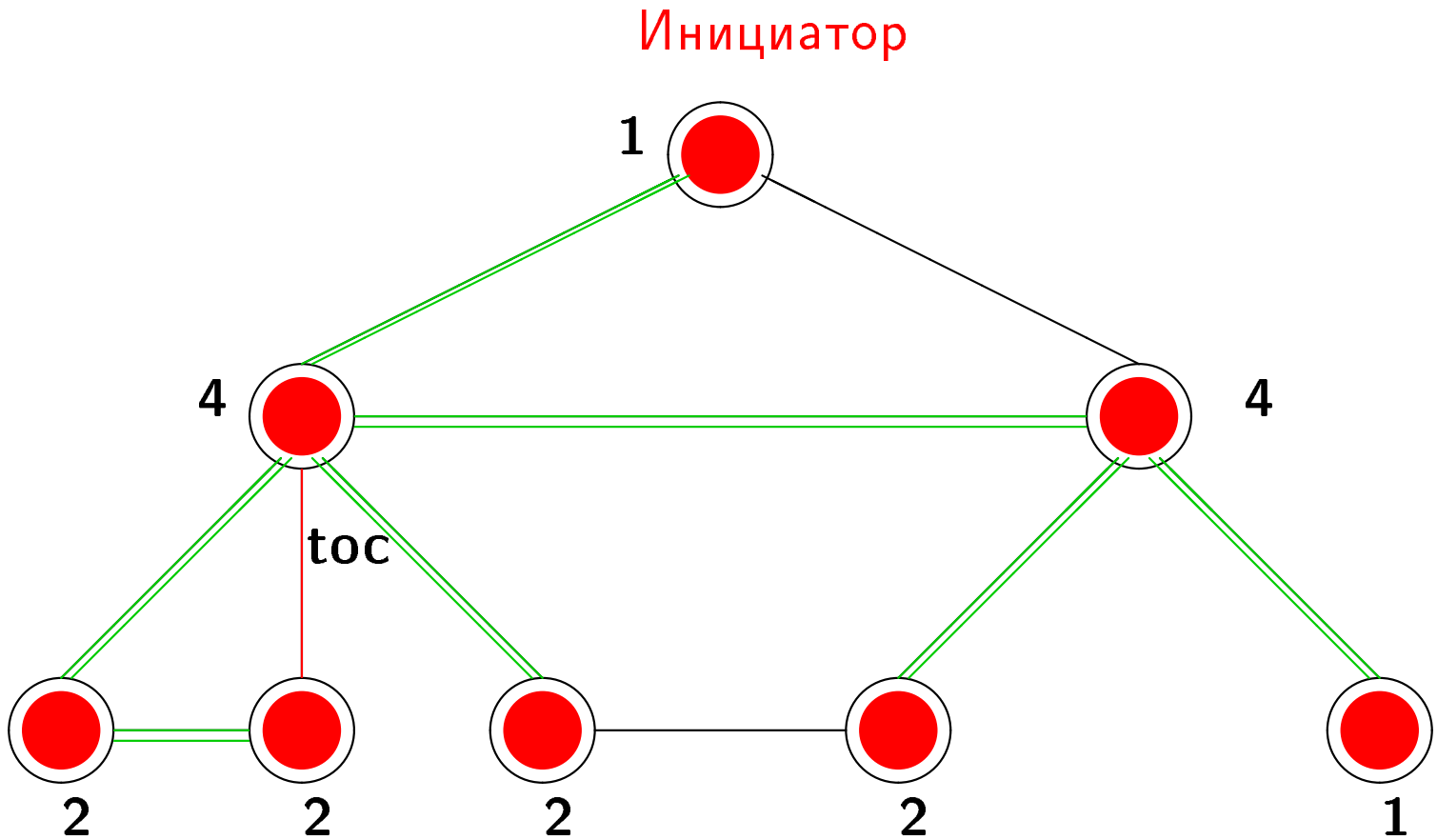
Алгоритм эха



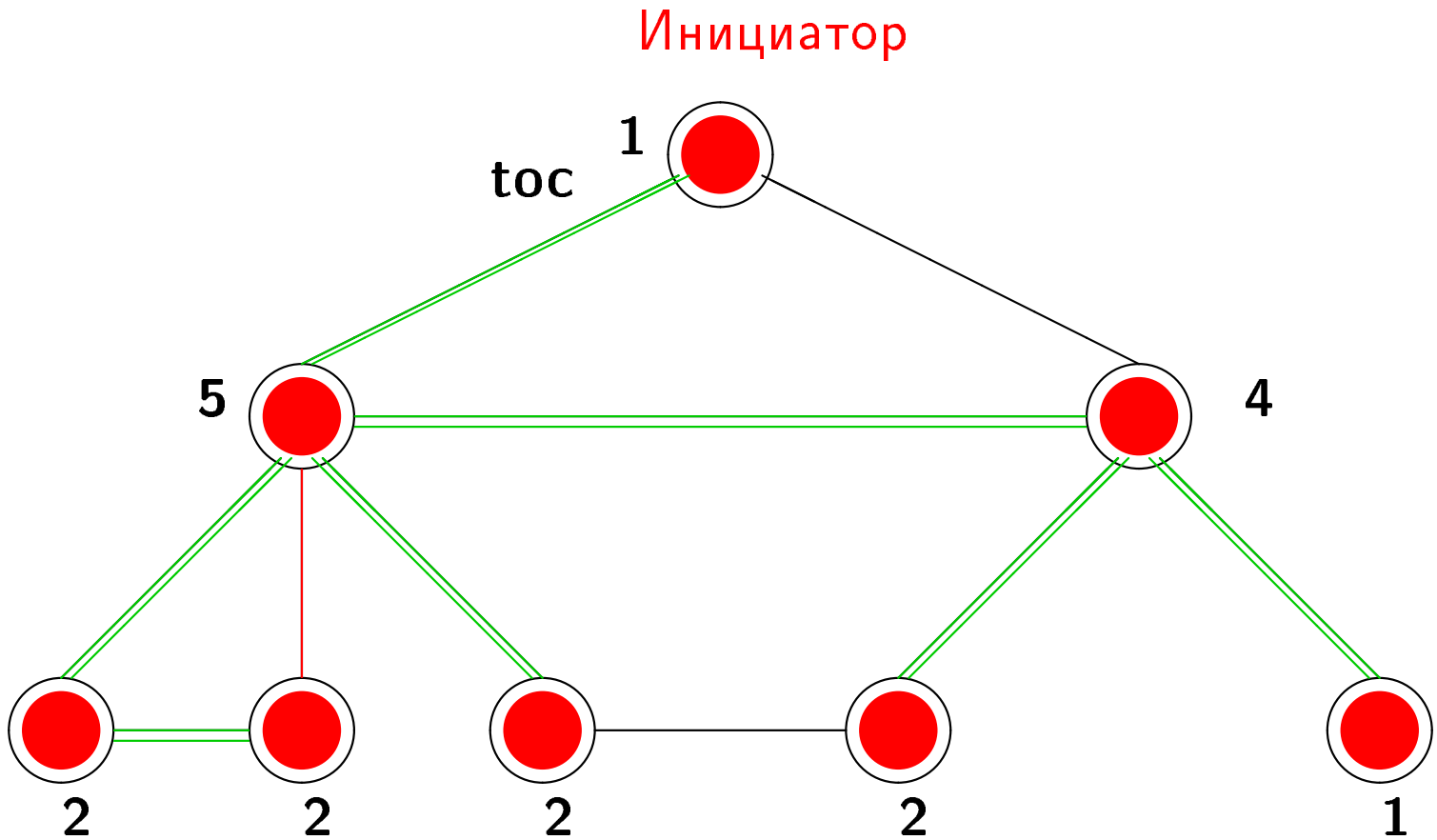
Алгоритм эха



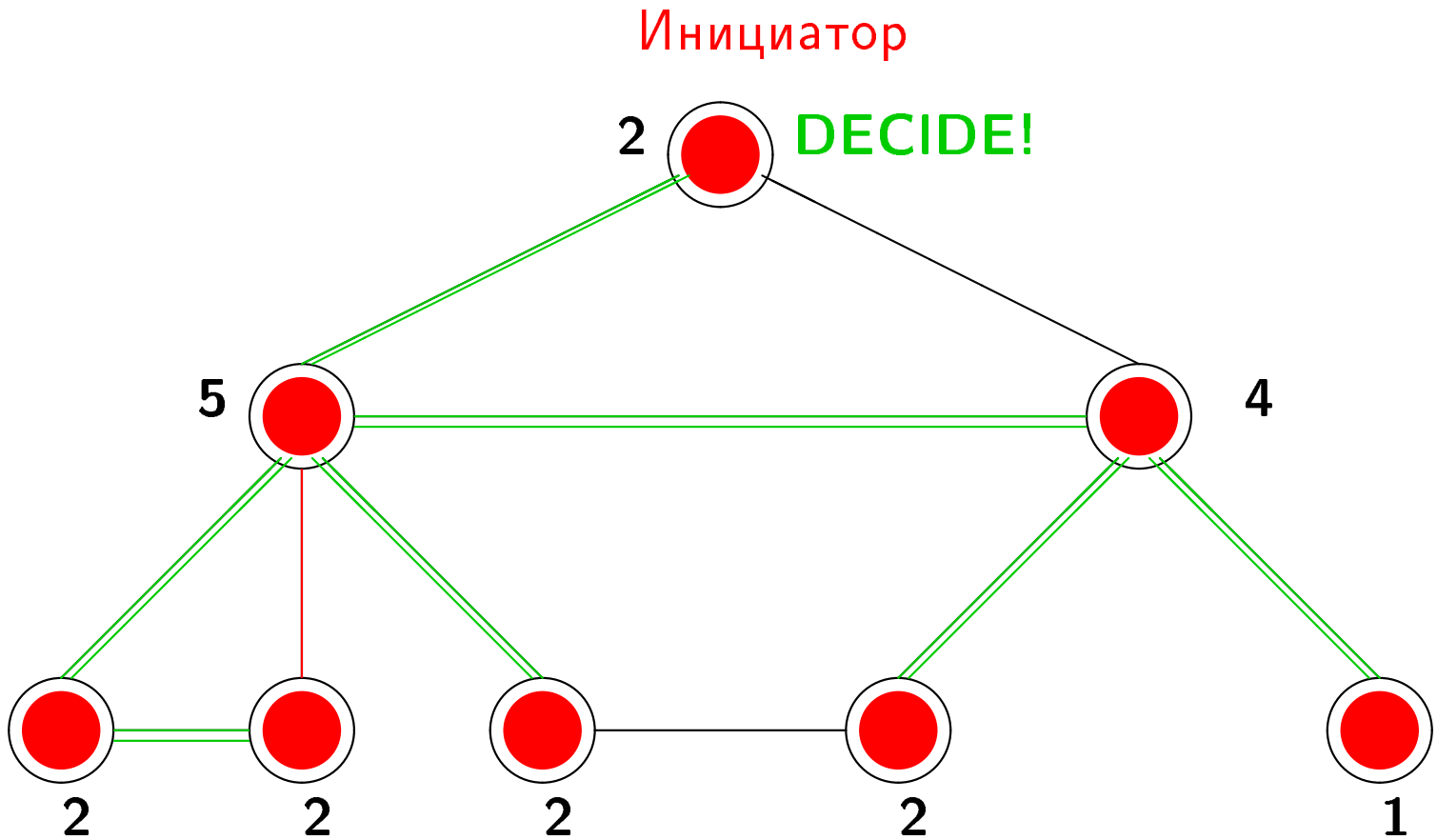
Алгоритм эха



Алгоритм эха



Алгоритм эха



Алгоритм эха

Утверждение 14.

Алгоритм эха является волновым алгоритмом.

Алгоритм эха

Утверждение 14.

Алгоритм эха является волновым алгоритмом.

Доказательство

Задача для самостоятельного решения.

Задачи на дом

Задача 7.

Приведите пример PIF-алгоритма для систем с *синхронным* обменом сообщениями, который не позволяет проводить вычисление точных нижних граней

Задача 8.

Покажите, что в каждом вычислении древесного алгоритма в точности два процесса принимают решение.