コンピュータサイエンス基礎論, 2009年度

分散アルゴリズム入門

情報科学研究科コンピュータサイエンス専攻
角川　裕次

もくじ

1. 分散システムと分散アルゴリズム

分散システム概要

分散システムとは

通信ネットワークで結ばれた計算機群

- 計算機は地理的に分散
- 各計算機はネットワークを通じて通信
- さまざまな種類の計算機や通信路が混在

分散システムの目的

- 情報の交換 (Email, WWW など)
- 資源の共有 (プリンタ, スーパーコンピュータなど)
- 多重化による信頼性の向上
- 並列化によるパフォーマンス向上
- 機能分割によるシステム構造の単純化

分散システム設計上の諸問題

通信プロトコル
- ❖ メディアアクセス制御, エラー訂正, 経路制御など

セキュリティ
- ❖ 暗号, 認証, など

負荷分散
- ❖ Grid, クラスタなど

信頼性, 耐故障性
- ❖ 多重化, 障害検知など

動的変化適応性
- ❖ モバイルアドホックネットワーク, Peer-to-Peer ネットワーク, センサネットワークなど

並列システムと分散システムの違い

並列システム

- 1台の並列計算機, あるいは PC クラスタなど
- 各プロセッサが同期して並列動作
- システムの全貌は既知
- ある特定のプロセッサが計算を始動
- 計算の入力は特定の場所に集まっている

分散システム

- 様々な計算機が非同期に動作
- システムの全貌すら不明
- 動作を開始するプロセッサとその数が予め分からない
- 計算の入力は完全に分散

プロセッサ, ノード, プロセス ...

プロセッサ, ノード

- 物理的なもの

プロセス

- 計算実体の抽象化した論理的なもの
- 1つのプロセッサ上に複数のプロセスが存在しうる

分散アルゴリズム

- プロセスをその対象として考える
- 論理的な分散システム

以降、用語「プロセス」を使用

分散システムのモデル化

実際の分散システムとモデル化

実際の分散システム

- ❖ 計算機(プロセス) ： 一様 / 多種多様
- ❖ 通信リンク ： 一様 / 多種多様

都合の良いモデル上でアルゴリズムを考えたい

- ❖ 汎用性の高いモデルが好ましい
  - ✔ より様々なシステムで動作可能
- ❖ 汎用性が高いと問題が解けない場合があるので要注意

モデルの例 ： 非同期システム

プロセスの実行速度

- ❖ 実行速度は一定とは限らない

通信リンク

- ❖ メッセージによる通信
- ❖ メッセージ伝送時間は有限だが上限の保証なし

時計

- ❖ 時間の進み具合はプロセスごとに全くバラバラ
- ❖ システム共通の時刻は仮定できない

モデルの例 : プロセスの種類と数

対等なプロセス

- ❖ 特別なプロセスは存在しない
- ❖ 各プロセスは同一の分散アルゴリズムを実行

任意の参加プロセス数

- ❖ プロセス数はパラメータ N として表記
- ❖ 各プロセスは必ずしも N の値を知っているとは限らない

モデルの例 ： アルゴリズムの始動

始動プロセスと非始動プロセス

- ❖ **始動プロセス**は自律的にアルゴリズムの実行を開始
- ❖ **非始動プロセス**はメッセージを受信して実行を開始

始動プロセス集合は前もっては分からない

- ❖ 任意の始動プロセス集合に対する正しさが必要

故障, 動的変化 等に関する諸モデル

古典的モデル

- プロセス集合と通信リンク集合は変らない・故障なし

故障モデル

- プロセスが停止故障を起こす場合がある
- 故障したプロセスは復活しない

動的モデル

- プロセス集合と通信リンク集合は時々刻々変る
- {Peer-to-peer, モバイルアドホック}ネットワーク

分散アルゴリズムの難しさ

非同期性

- ❖ 各プロセスは自律的に並行に動作 (始動プロセスが複数)
- ❖ 各プロセスの動作タイミングはバラバラ
- ❖ メッセージ伝達に時間がかかる

状態観測が困難

- ❖ システム全体の状態を瞬時には調査できない
- ❖ システムの一部を調べている間に他所が変化

分散アルゴリズム評価尺度

(通常の)逐次アルゴリズム

時間複雑度

- ❖ 動作完了にかかる実行ステップ数

空間複雑度

- ❖ 動作の間に使用する記憶領域や内部状態数

分散アルゴリズムの評価尺度 1

評価尺度: メッセージ複雑度

- ❖ アルゴリズム実行中に
  プロセス間で交換されるメッセージ総数
- ❖ 1メッセージは $O(\log N)$ ビットと仮定
  - ✔ 定数個のプロセス識別子をメッセージに保持可能

考え方の基本: メッセージ送信回数が実行時間を支配

- ❖ プロセッサは非常に高速
- ❖ ネットワークはそれほど速くない

メモ:

- ❖ 送信された総ビット数で測る場合もあり

分散アルゴリズムの評価尺度 2

評価尺度: 時間複雑度

- ❖ アルゴリズム終了までのラウンド数
- ❖ ラウンド ： 受信、計算、送信の1サイクル

評価尺度: 空間複雑度

- ❖ 各プロセスが用いるメモリの量

2. 分散アルゴリズム研究で取り扱う諸問題

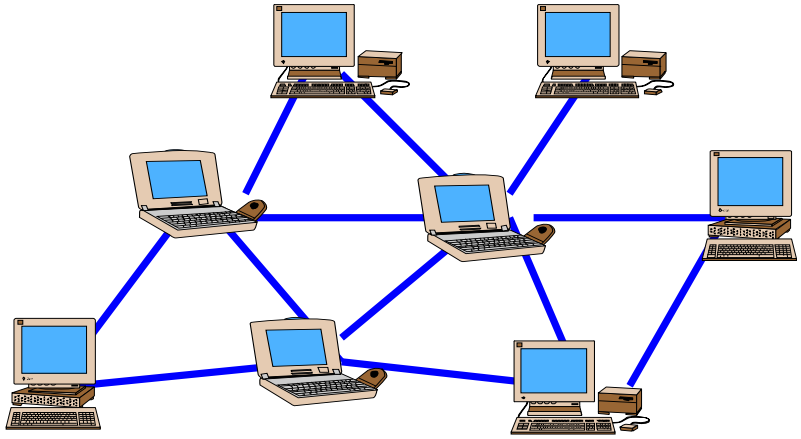
問題の例 (その1)

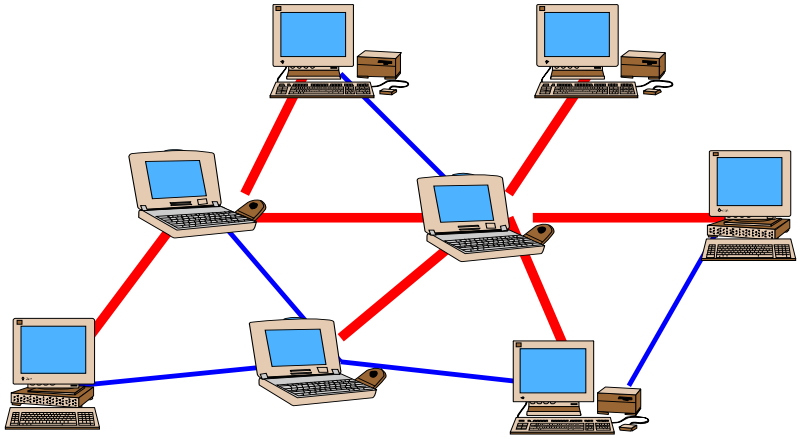
生成木構成

- ❖ 分散システムの通信ネットワークをグラフと考え、
  そのグラフ上での(最小)生成木を求める
- ❖ オーバーレイネットワーク構成の一種
- ❖ 応用: 通信コストの最小化

注意 : 各プロセスの入出力

- ❖ 入力 : 隣接プロセス集合
  - ✔ 各プロセスにグラフが入力として与えられるのではない
- ❖ 出力 : 接続リンクより生成木(の一部)を構成するリンク





問題の例 (その2)

リーダー選挙

- 1つのプロセスを「リーダー」に選出
- 応用: リーダーはシステム全体のコーディネーター

相互排除

- ある特定の動作が許可されるプロセスを1つに限定
- 動作を行なうプロセスは時々刻々変わってゆく
- 応用: 共有ファイルの更新、分散データベースの更新

問題の例 (その3)

大域スナップショット

- ❖ 分散システム全体の状態を記録するもの
- ❖ 各プロセスの状態, 伝送中のメッセージなど
- ❖ 応用: 故障からの復旧、分散システムのデバッグ

終了検出

- ❖ 分散アルゴリズムの動作が終了したか否かを検出
- ❖ 応用: システムの停止の検出、アルゴリズムの再実行

デッドロック検出

- ❖ 互いに待機し合うことによる動作の永久停止の検出

問題の例 (その4)

合意

- すべてのプロセスが同一の値に合意するための方法
- 応用: 分散トランザクション実行, 複製サーバー, 多重化による高信頼性

放送

- あるプロセスからすべてのプロセスへ、情報を効率良く伝搬するための送信順序の決定方法
- 応用: 情報伝達

問題の例 (その5)

時刻同期

- ❖ 各プロセスの実時間時計を(ほぼ)同じ値に合わせる
- ❖ 応用: 実時間による事象の前後関係の決定

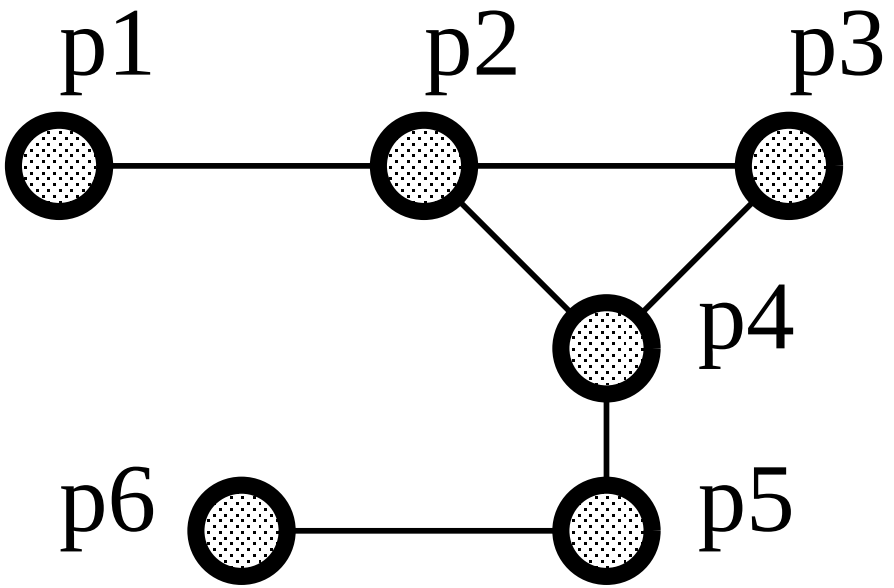
3. 形式的計算モデル

システムモデルとアルゴリズム

分散システムのグラフ表現

節点: プロセス

辺: 2プロセス間の通信リンク



プロセス

プロセスは状態機械

- P_i $(i = 1, 2, ..., N)$ と表記 (N はプロセス総数)
- 内部状態を持つ
  - ✔ メモリやCPUレジスタなどの抽象化

動作を3種類の事象(事象)に抽象化

- 内部(Internal)事象： プロセスの局所的な計算
- 送信(Send)事象： メッセージの送信
- 受信(Receive)事象： メッセージの受信

通信リンク: 2つのプロセス間の通信路

多くの場合 FIFO (First-In First-Out) **性を仮定**

- **先に送ったメッセージは先に到着**
- TCP

FIFO**性を仮定しないモデルもある**

- UDP

分散アルゴリズムの定義

分散アルゴリズム

- 各プロセス P_i の局所アルゴリズムの集合

各プロセス P_i の局所アルゴリズム $(Z, I, \vdash^i, \vdash^s, \vdash^r)$

- Z：プロセスの局所状態の集合
- I：初期局所状態の集合 (Z の部分集合)
- $\vdash^i$：内部事象による局所状態の変化規則
- $\vdash^s$：送信事象による局所状態の変化規則
- $\vdash^r$：受信事象による局所状態の変化規則

状況 (configuration)　(或は 大域状態 global state)

状況： 分散システム全体のある瞬間の状態を表現

状況は以下のものの組で表現

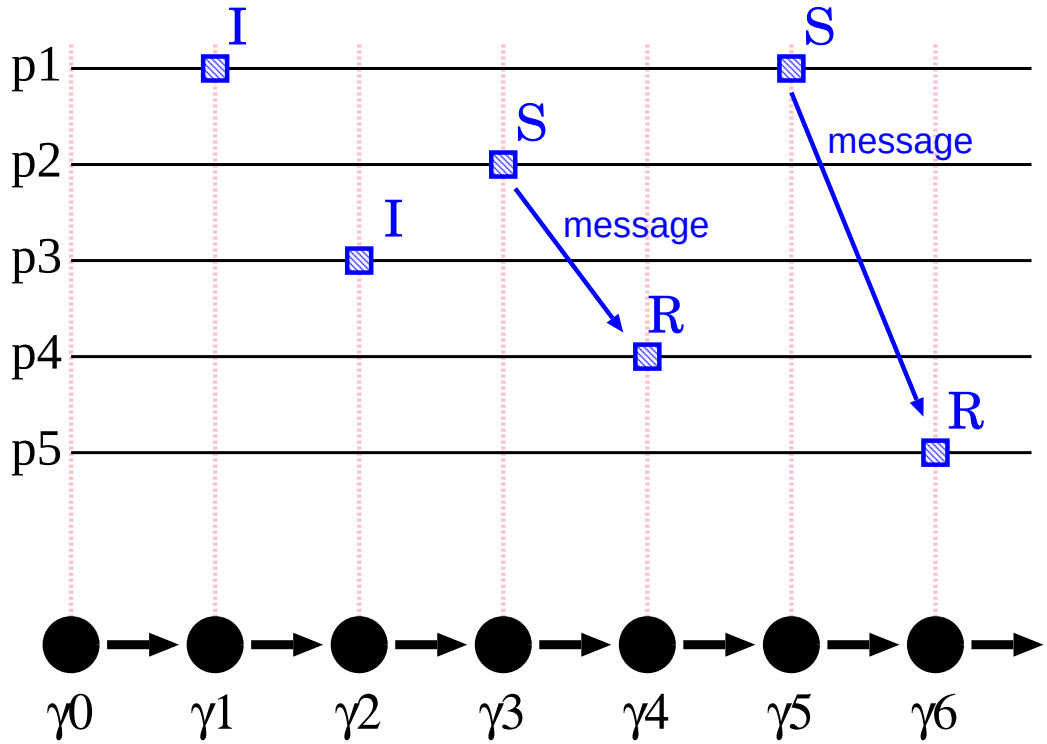
- ❖ 各プロセスの局所状態
- ❖ 各通信リンクの状態 (伝送途中メッセージの集合)

分散システムは状態遷移システムと見なせる

- ❖ 状況を状態とする
- ❖ 事象により状態遷移

時空図 (time-space diagram)

システムの動きを図示



- I : **内部**(Internal)**事象**
- S : **送信**(Send)**事象**
- R : **受信**(Receive)**事象**
- γ_i : **分散システムの状況**

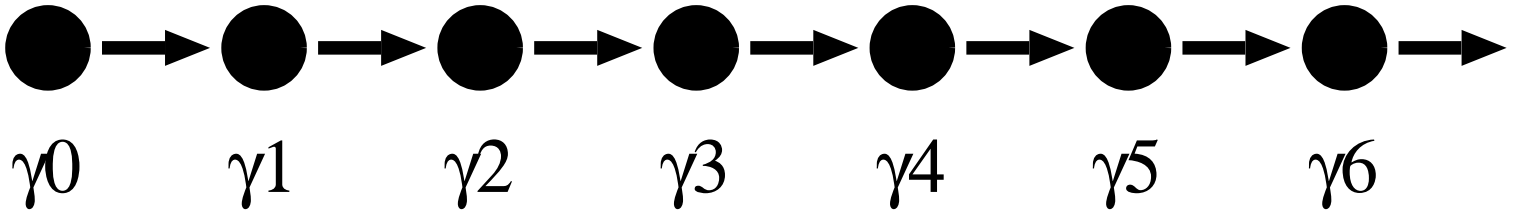
状態遷移システムとしての分散システム

状態遷移システム $S = (\mathcal{C}, \rightarrow, \mathcal{I})$

- $\mathcal{C}$：分散システムの可能な状況すべての集合
- $\rightarrow$：状態遷移規則 ($\mathcal{C} \times \mathcal{C}$ の部分集合)
- $\mathcal{I}$：可能な初期状況の集合 ($\mathcal{C}$ の部分集合)

状態遷移システム S の実行 E

- $E = (\gamma_0, \gamma_1, \gamma_2, \ldots)$ の極大列
- ただし $\gamma_i \in \mathcal{C},\ \gamma_0 \in \mathcal{I},\ \gamma_i \rightarrow \gamma_{i+1}$



仕様記述

安全性 (Safety)

システムにおいて常に成り立つべき性質

- あるいは発生しては困る性質の否定
- 動作開始時から常に成立

例

- システム内のトークン数はちょうど1つ
- 共有ファイルを更新するプロセスの数は高々1つ

生存性 (Liveness)

有限時間内に成り立つべき性質

- ❖ いつかは必ず成り立って欲しい性質
- ❖ 要求等がいつかは満足されることを記述

例

- ❖ いつかはトークンが自分プロセスに巡ってくる
- ❖ 要求を出せば応答がいずれ得られる

仕様の証明法, 安全性の証明

構造帰納法 (structual induction) を用いる方法

- Base Case:
  初期状況 γ_0 で成り立つことを示す
- Induction Hypothesis:
  任意の状況 γ_i で成り立つと仮定
- Induction Step:
  $\gamma_i \longrightarrow \gamma_{i+1}$ である任意のγ_{i+1} での成立を示す

仕様の証明法, 生存性の証明 (1)

背理法

- 未来永劫、生存性が満たされないと仮定
- そのような実行が可能か否かを検証
- 仮定がアルゴリズムの動作に矛盾することを示す

仕様の証明法, 生存性の証明 (2)

関数 F を見つけ出す

- $F(\gamma_i)$ が極小でないとき　(γ_i は状況)
  γ_i から始まるどの実行に対しても、ある γ_j が存在して
  $\gamma_i \longrightarrow \cdots \longrightarrow \gamma_j$ となり、$F(\gamma_i) > F(\gamma_j)$ が成立

- F の値が極小のとき: 証明したい性質が成立

せつめい: F はある種のポテンシャル場

- 系はポテンシャルの低い方へ移動して安定
- 最小ポテンシャルが望みの状況

時相論理 (temporal logic) による仕様記述

時相論理演算子 $\Box$

- 「**常に** (always)」

時相論理演算子 $\Diamond$

- 「**いつか** (eventually)」

例: $\Box(\text{Req} \longrightarrow \Diamond\text{Ack})$

- **もし要求すれば** (Req) **いずれ応答が返ってくる** (Ack)

例: $\Box \Diamond \text{Move}$

- **いずれ動作** (Move) **することが無限回起きる**

分散アルゴリズムのモデル検証 (Model Checking)

状態遷移システムの検証そのもの

- ❖ システムの状態遷移を記述
- ❖ 仕様を時相論理式 (temporal logic) で記述

システムが時相論理式を満たすか否かを検証

分散システムにおける時刻

発生した事象間の前後関係

前後 (happened-before) 関係

- ２事象間における発生の前後の関係

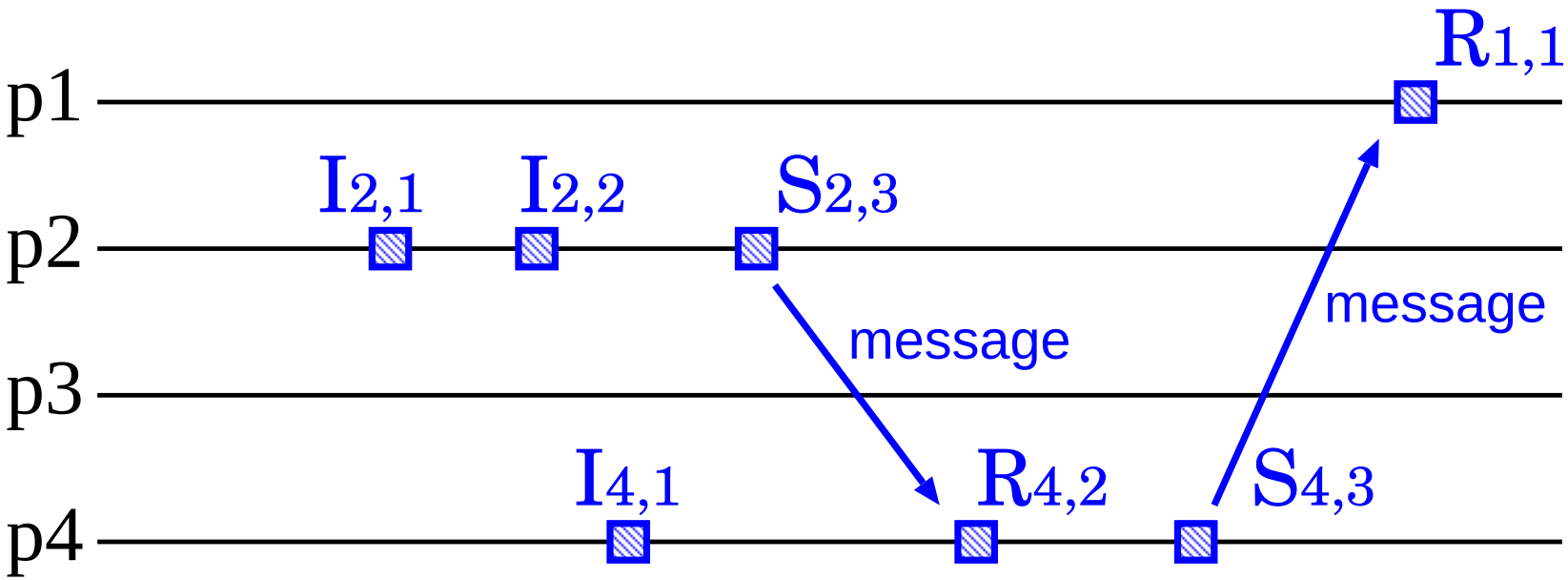
実現方法: 発生時刻による方法では駄目

- 非同期分散システムにはシステム共通の時計がない
- 「時刻」に基づく前後関係は意味をなさない

ほとんどの場合は事象間の因果関係が分かれば十分

- 因果関係があれば前後関係がある

事象間の前後関係



- $I_{2,2}$ は $S_{2,3}$ よりも先に発生
- $S_{2,3}$ は $R_{4,2}$ よりも先に発生
- 従って $I_{2,2}$ は $R_{4,2}$ よりも先に発生

前後関係：$a \prec b$ の定義

同一プロセスの内部事象間

- a, b ともに同一プロセスで発生し
  a は b よりも前に発生

同一メッセージの送受信事象間

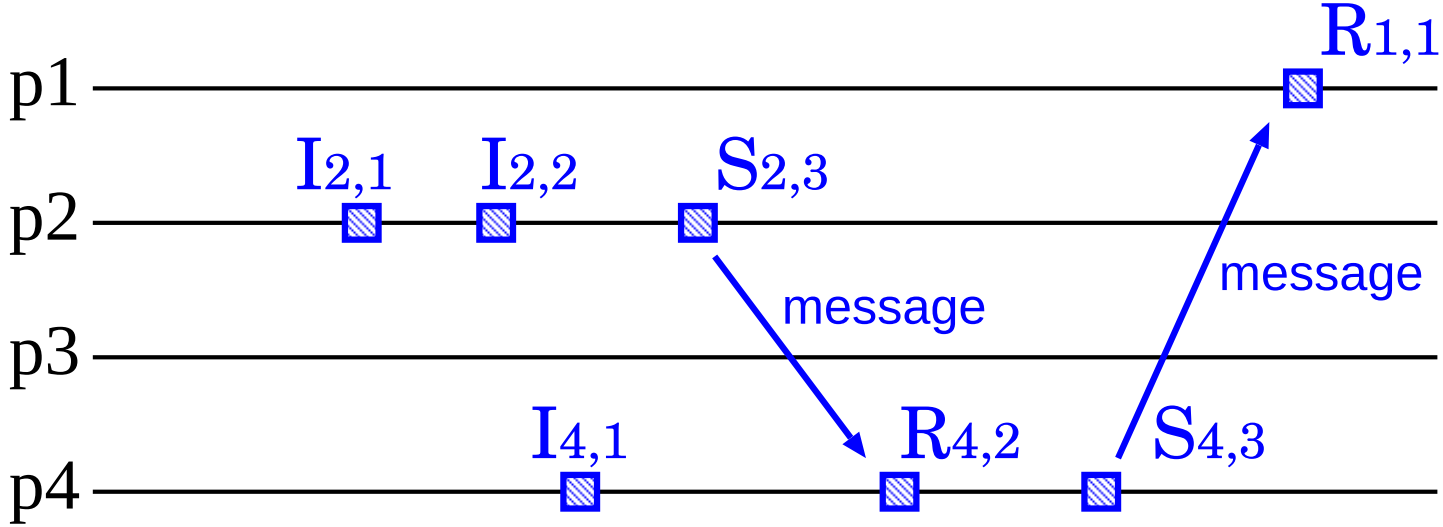
- あるメッセージに対し
  a は送信事象
  b は対応する受信事象

推移律

- $a \prec c$　かつ　$c \prec b$　のとき

関係 $\prec$ は半順序

すべての a, b に対し
必ずしも $a \prec b$ または $b \prec a$ とは限らない



並行(concurrent)

- **先でも後でもない**
- **$I_{2,2}$ と $I_{4,1}$ は並行**
- **$S_{2,3}$ と $I_{4,1}$ は並行**

論理的な時計の構成

事象間の前後関係の比較

- 各事象に時刻値を割り当てる
- 事象間に前後関係があれば対応する時刻値間に大小関係

時計関数 Θ

- $\Theta(a)$:　事象 a に対する時刻値
- 時刻値のデータ型: 整数やベクトルなど

注意

- 観察可能なものは時刻値
- 比較は時刻値間でおこなう
- 時刻値間の大小関係は事象間の前後関係と一致するとは限らない

時計関数 Θ の条件 (弱)

各事象 a,b に対し $(a \prec b) \Rightarrow (\Theta(a) < \Theta(b))$

説明

- 先に発生した事象の時刻値は
  後の事象の時刻値よりも小さい

- 逆は真とは限らない
  つまり $\Theta(a) < \Theta(b)$ のとき $a \prec b$ とは限らない

例: Lamport による論理時計

- このあと紹介

Lamport の論理時計

時計関数 Θ の値は整数

- 各事象 a, b に対し $(a \prec b) \quad \Rightarrow \quad (\Theta(a) < \Theta(b))$
- 比較 $<$ は整数に対する大小比較
- 異なる事象には異なる整数値

2つの事象が並行か否かが区別つかない

- 半順序関係の事象に対して時計値は全順序

Lamport の論理時計, 実装概要 (1)

各プロセス P_i は局所変数 c_i を持つ

- 初期値は 0

内部事象発生時

- c_i を 1 増やし、この値が事象の時刻印

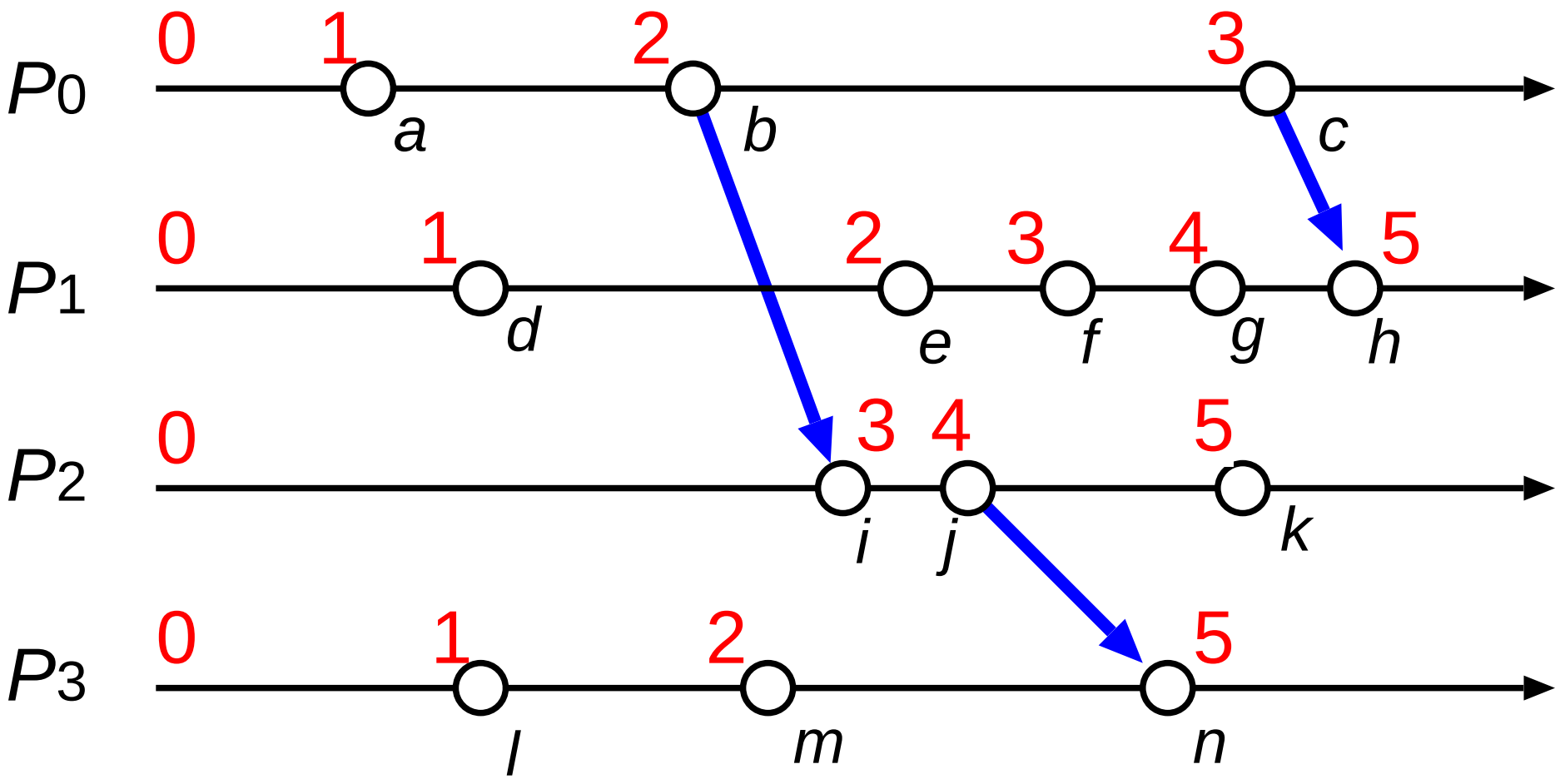
メッセージ送信時

- c_i を 1 増やし、この値が事象の時刻印
- メッセージに c_i を添付して送信

メッセージ受信時

- メッセージ添付の c_j を得る
- $c_i := \max(c_i, c_j) + 1$ で更新し、この値が事象の時刻印

Lamport の論理時計, 実装概要 (2)



Lamport の論理時計, 実装概要 (3)

P_i で発生した事象 a の時計値 $\Theta(a) = c_a N + P_i$

- ✔ 同じ時刻印でも異なるプロセスでは異なる時計値
  - ❖ c_a ： 事象 a の時刻印
  - ❖ N ： プロセス総数
  - ❖ プロセス識別子 ： $0 \leq P_i < N$

時計値は事象ごとに互いに異なる　(全順序)

- ❖ 整数値として大小比較が可能

$(a \prec b) \quad \Rightarrow \quad (\Theta(a) < \Theta(b))$　を保証

- ❖ 逆は真とは限らない

時計関数 Θ の条件 (強)

事象 a, b に対し

$$(a \prec b) \quad \Leftrightarrow \quad (\Theta(a) < \Theta(b))$$

説明

- 前後関係と時計値の大小が同値
- 並行な2つの事象の時計値は比較不能
  - ✔ 「比較不能」 = 「大小どちらも不成立」

- 時計値の比較で事象間の前後／並行が分かる

例: ベクトル時計

ベクトル時計 : 概要

時計関数 Θ の値はベクトル

- **整数を要素とする大きさ N (プロセス総数) のベクトル**

各プロセス P_i が持つベクトル V_i :

- **第i要素 $V_i[i]$ は P_i の時刻印**
- **第j要素 $V_i[j]$ $(i \neq j)$ は P_i の知る最新の P_j の時刻印**

2つのベクトル間の大小比較 $<$:

- **ベクトル間の対応する要素全てに対し$<$の関係**

詳細は省略

4. いくつかの分散アルゴリズム

リーダー選挙問題

リーダー選挙問題

プロセス群の中から1つのプロセスを選出

- 選出プロセスが**リーダー**

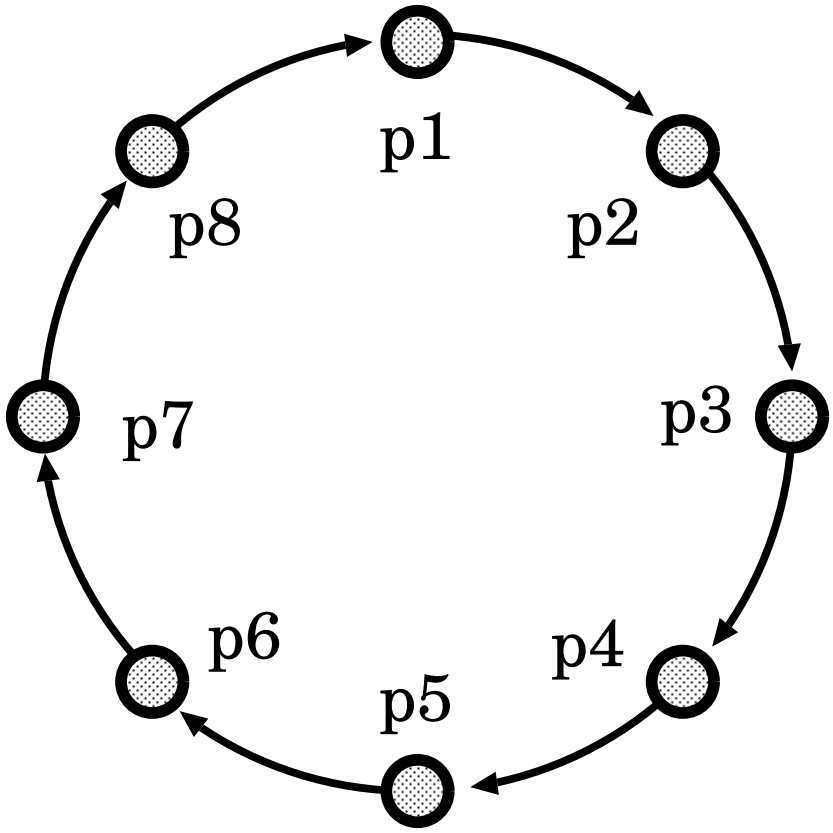
定式化

- 各プロセス p_i は変数 state_i を持つ
- 選出されたプロセスは$\mathrm{state}_i = \mathrm{leader}$ に設定
- それ以外プロセスは$\mathrm{state}_i = \mathrm{lost}$ に設定

仮定

- 各プロセスは識別子(ネットワークアドレス)を持つ

ネットワークは単方向リング



- ❖ 矢印は通信リンク
- ❖ 矢印の方向へのみメッセージを送信可能

2つのアルゴリズム

LeLann のアルゴリズム

- メッセージ複雑度は $O(N^2)$
- アルゴリズムは単純

Chang & Roberts のアルゴリズム

- LeLann のアルゴリズムの改良版
- メッセージ複雑度は $O(N^2)$
- 期待値では $O(N \log N)$

LeLann のアルゴリズム

LeLann のアルゴリズムの基本方針

リーダー選出条件

- ❖ リーダー: 始動プロセス中でプロセス識別子が最小
- ❖ 非始動プロセスはリーダーにはならない

アルゴリズムの方針

- ❖ 各始動プロセスのプロセス識別子を巡回させる
- ❖ 他の始動プロセスの識別子を全て知ることができる

始動プロセスのなかの最小識別子を決定可能

- ❖ 最小識別子が自分の識別子に等しければリーダになる
- ❖ そうでなければ非リーダーとなる

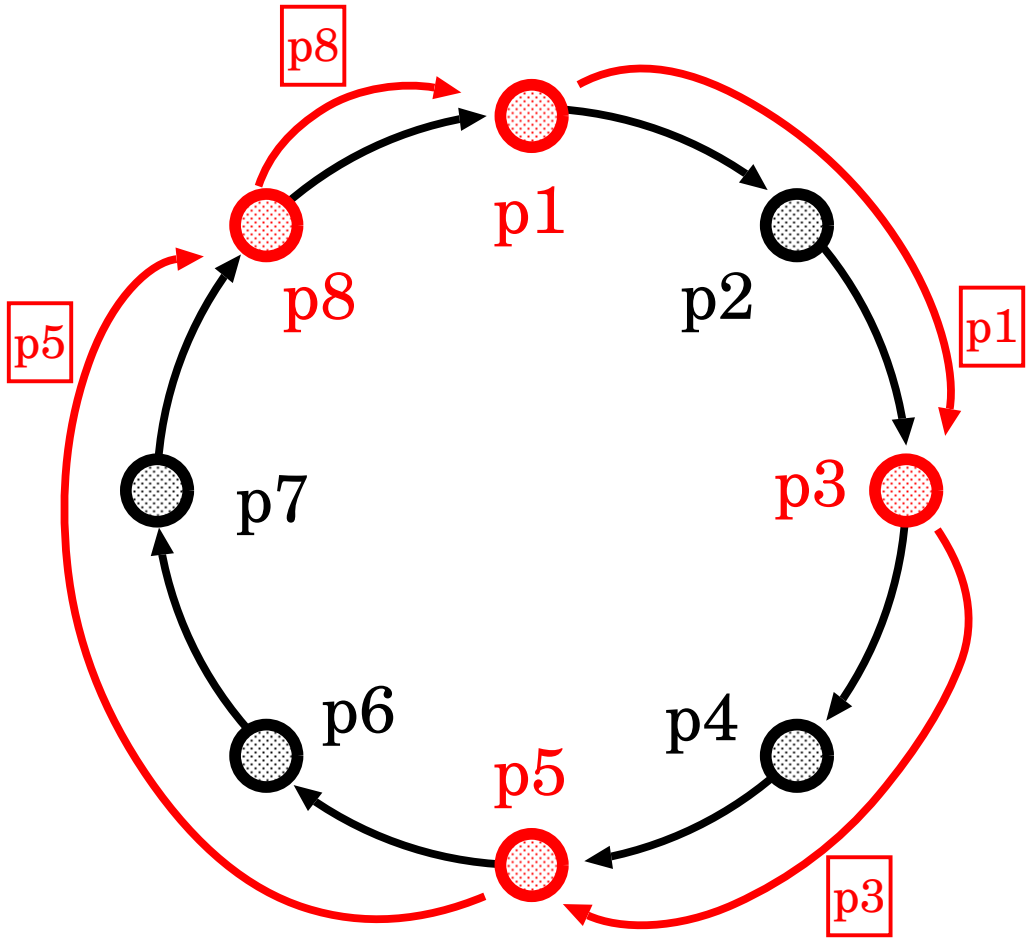
非始動プロセスの動作

自分は非リーダーと決定

ひたすらメッセージ転送...

- ❖ メッセージを隣りから受けとり、別の隣へ転送
- ❖ これを繰り返す

動作のイメージ図



各始動プロセスの識別子を送信した様子
(このあと受信し、転送…)

始動プロセス P_i の動作 ： 概要

局所変数: List_i

- ❖ これまでに観察したプロセス識別子の集合を蓄える
- ❖ 初期値として自分の識別子 P_i を入れる

1. 自分の識別子 P_i を左隣のプロセスへ送信;
2. 右隣から P_x を受信;
3. If ($P_x =$ 自分の識別子) Then Goto 7;
4. P_x を変数 List に追加;
5. P_x を左隣のプロセスへ送信;
6. Goto 2;
7. List_i 中の最小識別子が P_i に等しいか否かで
   P_i がリーダーか否かを判定;

LeLann のアルゴリズム: プロセス P_i の動作 (1/3)


```
1   var   List_i : ℙ — 初期値 {P_i};
2         state_i : (sleep, leader, lost, cand) init sleep;
3   if (P_i は非始動プロセス) {
4         state_i = lost;
5         ※非始動プロセスの動作
6   } else {
7         state_i := cand;
8         ※始動プロセスの動作
9         if (P_i = min(List_i))   state_i := leader
10        else                     state_i := lost
11  }
```

LeLann のアルゴリズム: プロセス P_i の動作 (2/3)

※非始動プロセスの動作

while (true) {

メッセージ $\langle \text{tok}, P_x \rangle$ **を受信**;

隣接プロセスへ $\langle \text{tok}, P_x \rangle$ **を送信**;

}

LeLann のアルゴリズム: プロセス P_i の動作 (3/3)

※始動プロセスの動作

隣接プロセスへ $\langle \text{tok}, P_i \rangle$ **を送信**;

メッセージ $\langle \text{tok}, P_x \rangle$ **を受信**;
while $(P_x \neq P_i)$ { — **自分の識別子** P_i **が一巡するまで**
  $List_i := List_i \cup \{P_x\}$; — **観察した識別子の蓄積**
  隣接プロセスへ $\langle \text{tok}, P_x \rangle$ **を送信**; — **識別子の転送**
  メッセージ $\langle \text{tok}, P_x \rangle$ **を受信**;
}

メッセージ複雑度解析, 最悪時

最悪時は全プロセスが始動プロセスのとき

- N 個のプロセスが識別子を送信
- 各識別子がネットワークを一巡 (それぞれ N 回送信)
- メッセージ総数は N^2

メッセージ複雑度解析, 最良時

最良時は始動プロセスが1つだけのとき

- ひとつのプロセスがプロセス識別子を送信
- ネットワークを一巡してアルゴリズムが終了
- メッセージ総数は N

Chang & Roberts のアルゴリズム

LeLann のアルゴリズムの欠点

- 実行途中でリーダになり得ない識別子が分かる
- それでも転送する
- メッセージ数が増える原因に

Chang & Roberts のアルゴリズム

- リーダーになり得ない識別子は転送しない (識別子破棄)
- リーダーの可能性のある識別子だけを転送
- 最小識別子だけがネットワークを最後まで巡回できる

自己安定分散アルゴリズム

自己安定 (Self-Stabilization) とは

分散システムの故障耐性のひとつ

- 故障: メモリ内容の変化, メッセージの消失, 内容変化等
- いわゆる「一時故障」(transient faults)

任意の初期状況から動作しても正しい状況に復帰

- 故障が発生してもいずれ正常動作に復帰

※通常の分散アルゴリズム

- ある決まったシステムの初期状況から動作を開始
  - ✔ 変数には初期値
  - ✔ プログラムカウンタの初期値は0
  - ✔ 通信リンクにはメッセージは流れていない

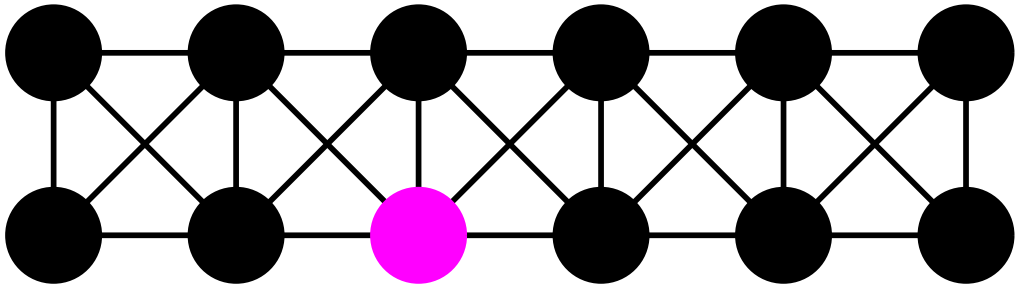
生成木問題 (1/2)

ネットワークに対する幅優先探索木(BFS)の計算

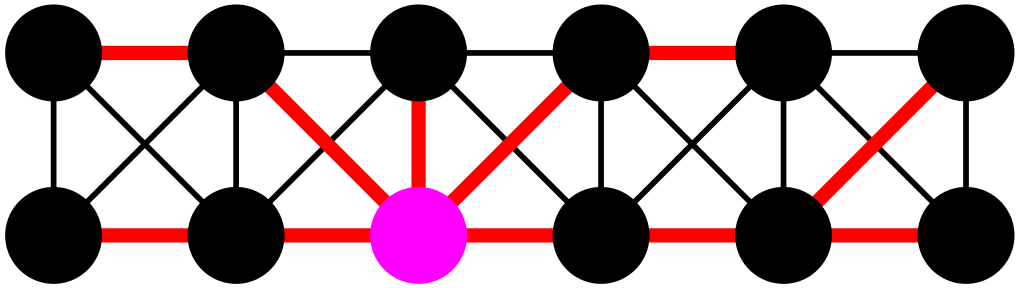
根プロセス(コーディネータ)が1つ存在

- ❖ 根プロセスだけ特別な動作が可能
- ❖ 他のプロセスは同一

ネットワーク

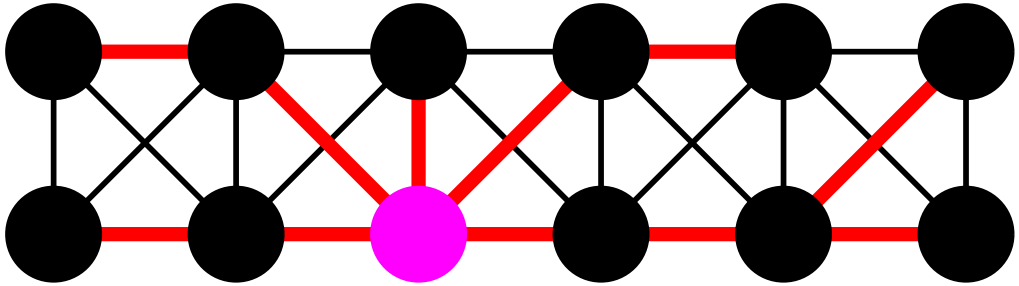


解



生成木問題 (2/2)

ネットワークに対する幅優先探索(BFS)木の計算



各プロセスの入力

- N_1 ： 隣接プロセスの集合

各プロセスの出力

- $f_i \in N_i$ ： BFS木での親プロセス

入力は分散、出力も分散

計算モデル

通信 : 局所共有変数モデル

- ❖ 隣接するプロセスの局所変数を参照可能
- ❖ 参照は瞬時に可能 (遅延なし)

根プロセスの存在を仮定

- ❖ 根プロセスだけ特別な動作を行う

自己安定な生成木アルゴリズム

各プロセス P_i の変数

- f_i : 親プロセス (出力)
- d_i : 根プロセスからのホップ数を保持 (作業用)

根プロセス P_0 の動作

- $d_0 := 0$; $f_0 := P_0$; を繰り返す

他のプロセス P_i の動作

- d_j が最小の $P_j \in N_i$ を発見
- $d_i := d_j + 1$;
- $f_i := P_j$;
- 以上を繰り返す

正しさ

根プロセス P_0 が d_0, f_0 の値を正しく設定

根プロセスの隣接プロセス P_i
- ❖ 根プロセスを親に選び、d_i, f_i の値を正しく設定

その他のプロセス
- ❖ 根プロセスを中心にして
  d_i, f_i の値を正しく設定するプロセスが広がる

… より詳しい動作
- ❖ d_i や f_i が正しくないプロセスたちは
  互いに d_i の値を上昇させてゆく
- ❖ いずれは根プロセスに近いプロセスを親として選択、
  d_i, f_i の値が正しくなる

おわり