離散最適化基礎論 (2025 年後学期)

高速指数時間アルゴリズム

第 7 回

包除原理 (1) : 原理

岡本 吉央　(電気通信大学)

okamotoy@uec.ac.jp

2025 年 11 月 25 日

最終更新：2025 年 11 月 26 日　07:18

スケジュール (前半)

1. 高速指数時間アルゴリズムの考え方 (10/7)
* 休み (体育祭) (10/14)
2. 分枝アルゴリズム：基礎 (10/21)
3. 分枝アルゴリズム：高速化 (10/28)
4. 分枝アルゴリズム：測度統治法 (11/4)
5. 動的計画法：基礎 (11/11)
6. 動的計画法：例 (11/18)

スケジュール (後半)

7. **包除原理：原理** (11/25)
* 休み (秋ターム試験) (12/2)
8. 包除原理：例 (12/9)
9. 部分集合たたみ込み：原理 (12/16)
* 休み (出張) (12/23)
* 休み (冬季休業) (12/30)
10. 部分集合たたみ込み：例 (1/6)
11. 指数時間仮説：原理 (1/13)
12. 指数時間仮説：証明 (1/20)
13. 最近の話題 (1/27)
* 休み (修士論文発表会) (2/3)

本日の内容

1. **包除原理とは**
2. 完全マッチングの数え上げ
3. ハミルトン路の数え上げ

ほうじょげんり

包除原理 = inclusion-exclusion principle

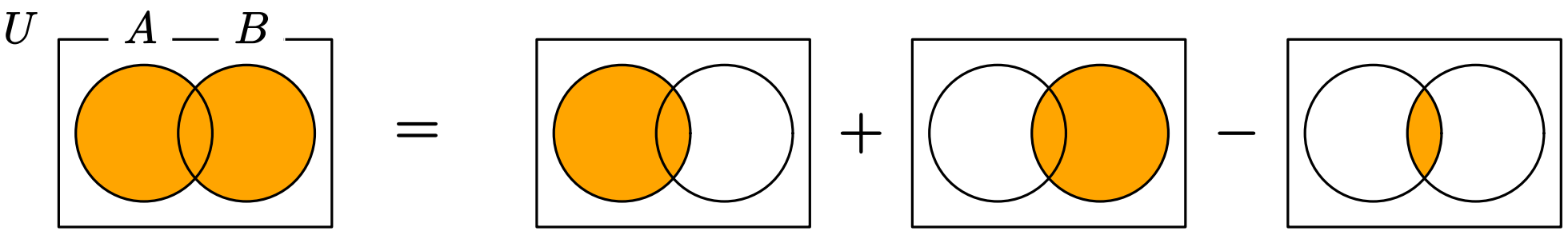
含める　除く

包除原理：簡単な場合 (1)

有限集合 $A, B \subseteq U$

包除原理

$$|A \cup B| = |A| + |B| - |A \cap B|$$

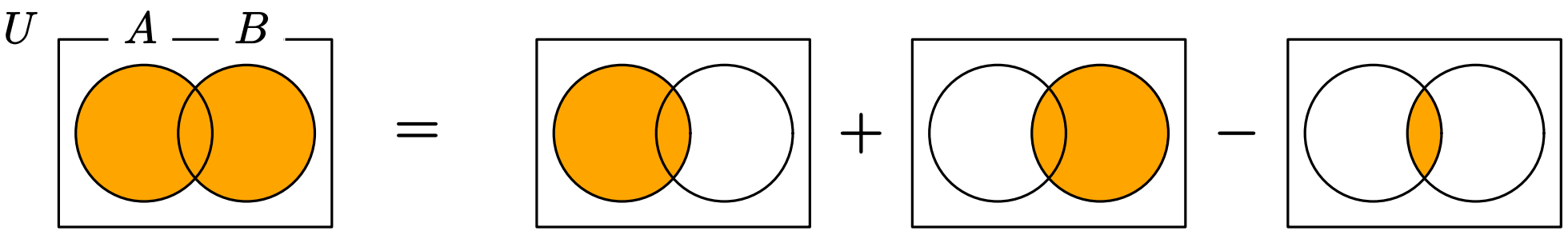


包除原理：簡単な場合 (1)

有限集合 $A, B \subseteq U$　　$X \subseteq U$ に対して，$\overline{X} = U - X$

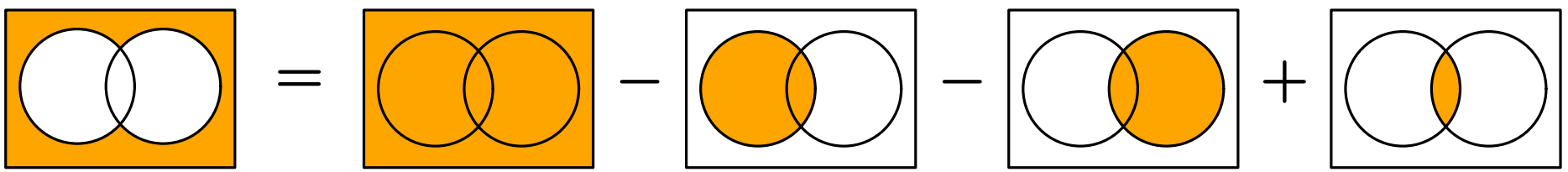
包除原理

$$|A \cup B| = |A| + |B| - |A \cap B|$$



包除原理 (補集合バージョン)

$$\left|\overline{A \cup B}\right| = |U| - |A| - |B| + |A \cap B|$$

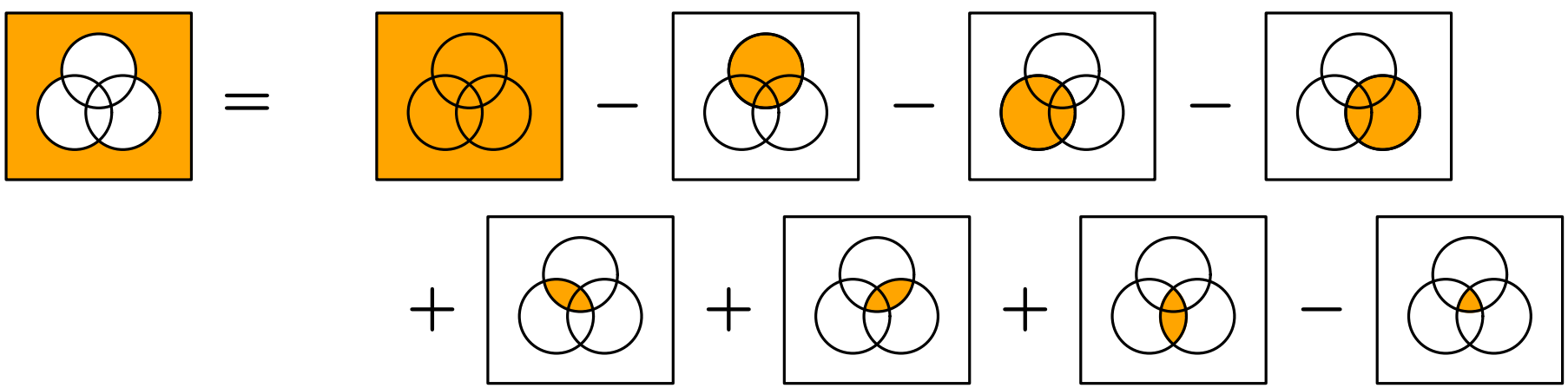


包除原理：簡単な場合 (2)

有限集合 $A_1, A_2, A_3 \subseteq U$

包除原理 (補集合バージョン)

$$\left|\overline{A_1 \cup A_2 \cup A_3}\right| = |U| - |A_1| - |A_2| - |A_3| + |A_1 \cap A_2| + |A_1 \cap A_3| + |A_2 \cap A_3| - |A_1 \cap A_2 \cap A_3|$$



包除原理：簡単な場合 (2) 続

$S \subseteq \{1,2,3\}$ に対して，$A_S = \bigcap_{i \in S} A_i$ とする

包除原理 (補集合バージョンの書き換え)

$$\left|\overline{A_1 \cup A_2 \cup A_3}\right| = |A_\emptyset| - |A_{\{1\}}| - |A_{\{2\}}| - |A_{\{3\}}| + |A_{\{1,2\}}| + |A_{\{1,3\}}| + |A_{\{2,3\}}| - |A_{\{1,2,3\}}|$$

包除原理：簡単な場合 (2) 続

$S \subseteq \{1,2,3\}$ に対して，$A_S = \bigcap_{i \in S} A_i$ とする

包除原理 (補集合バージョンの書き換え)

$$\left|\overline{A_1 \cup A_2 \cup A_3}\right| = |A_\emptyset| - |A_{\{1\}}| - |A_{\{2\}}| - |A_{\{3\}}| + |A_{\{1,2\}}| + |A_{\{1,3\}}| + |A_{\{2,3\}}| - |A_{\{1,2,3\}}|$$

$$= \sum_{S \subseteq \{1,2,3\}} (-1)^{|S|} |A_S|$$

包除原理：一般形

記法

- $A_1, A_2, \ldots, A_n \subseteq U$
- $[n] = \{1, 2, \ldots, n\}$
- 任意の $S \subseteq [n]$ に対して，$A_S = \bigcap_{i \in S} A_i$

定理：包除原理 (一般の n)

$$\left| \overline{\bigcup_{i \in [n]} A_i} \right| = \sum_{S \subseteq [n]} (-1)^{|S|} |A_S|$$

証明：n に関する帰納法 □

本日の内容

1. 包除原理とは
2. **完全マッチングの数え上げ**
3. ハミルトン路の数え上げ

- H.J. Ryser, *Combinatorial Mathematics*. The Carus Mathematical Monographs 14 (1963) MAA.

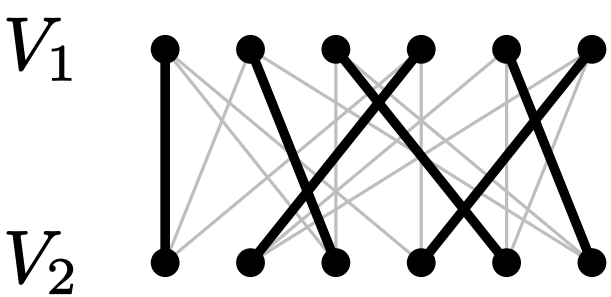
二部グラフの完全マッチング

二部グラフ $G = (V_1, V_2, E)$　(ただし，$V_1 \cap V_2 = \emptyset$)

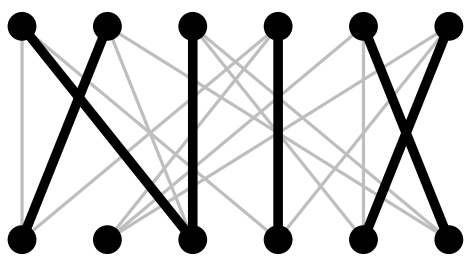
頂点集合が $V_1 \cup V_2$，辺集合が E で，
V_1 と V_2 の間にしか辺がない無向グラフ

定義：完全マッチング (perfect matching)

G の **完全マッチング** とは，辺部分集合 $M \subseteq E$ で
二部グラフ (V_1, V_2, M) にて各頂点の次数が 1 であるもの



完全マッチング である

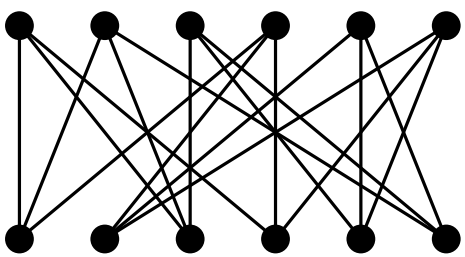


完全マッチング ではない

完全マッチングの数え上げ

問題：二部完全マッチングの数え上げ

入力： 二部グラフ $G = (V_1, V_2, E)$
出力： G の完全マッチングの総数

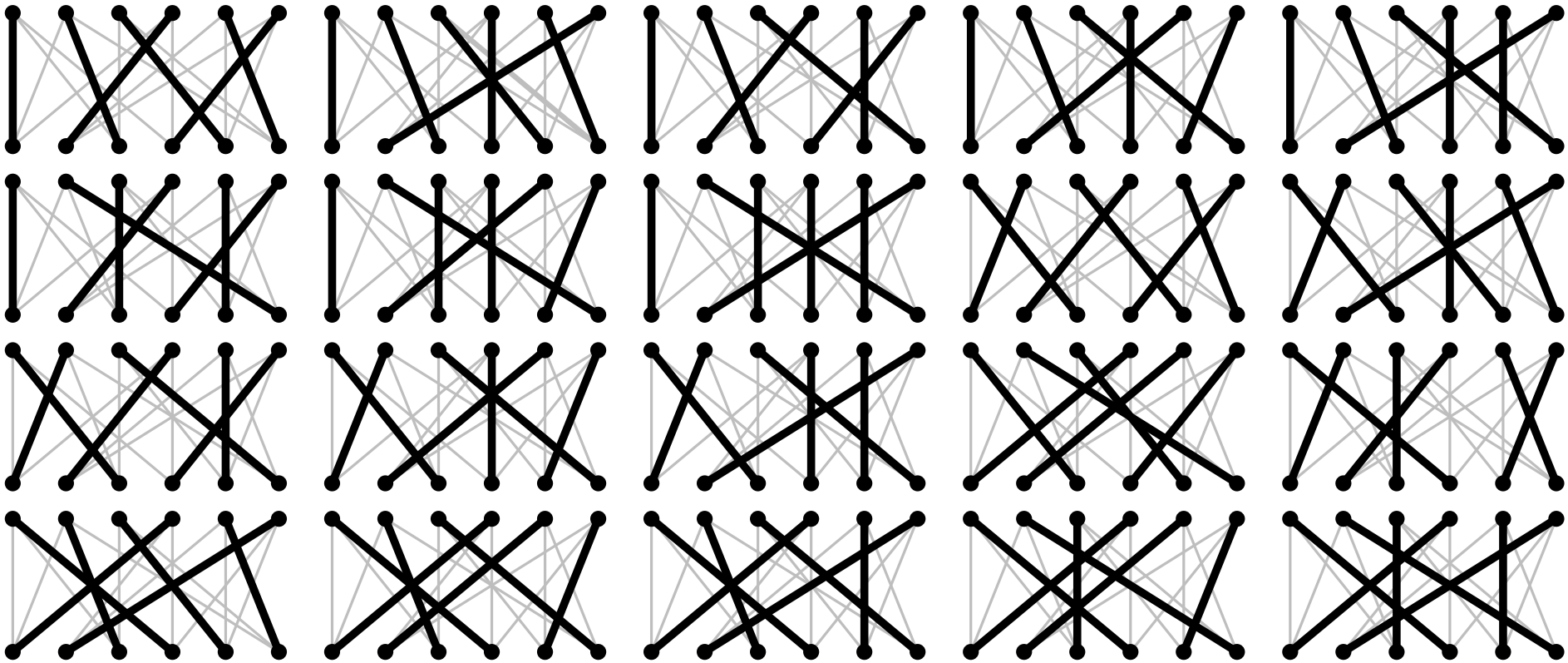


20

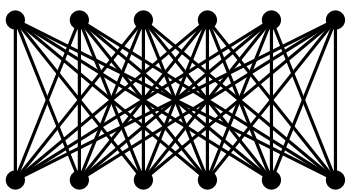
事実：二部完全マッチングの数え上げは #P 完全 (Valiant '79)

(注) #P 完全 $\Rightarrow$ NP 困難

しらみつぶし？



$|V_1| = |V_2| = n \quad \Rightarrow \quad$ 完全マッチングの総数 $\leq n!$



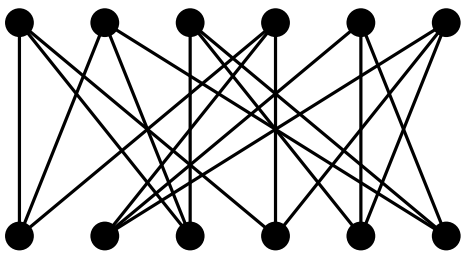
$\leftarrow \quad$ 完全マッチングの総数 $= n!$ となる場合

二部完全マッチングの数え上げ：包除原理

包除原理を使って，次を導く

定理：Ryser の公式 (’63)

二部完全マッチングの数え上げは $O^*(2^n)$ 時間でできる
$(n = |V_1| = |V_2|)$



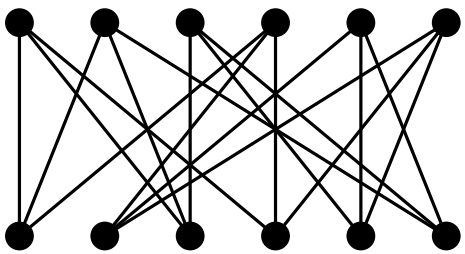
20

二部完全マッチングの数え上げ：包除原理

包除原理を使って，次を導く

定理：Ryser の公式 (’63)

二部完全マッチングの数え上げは $O^*(2^n)$ 時間でできる
$(n = |V_1| = |V_2|)$



20

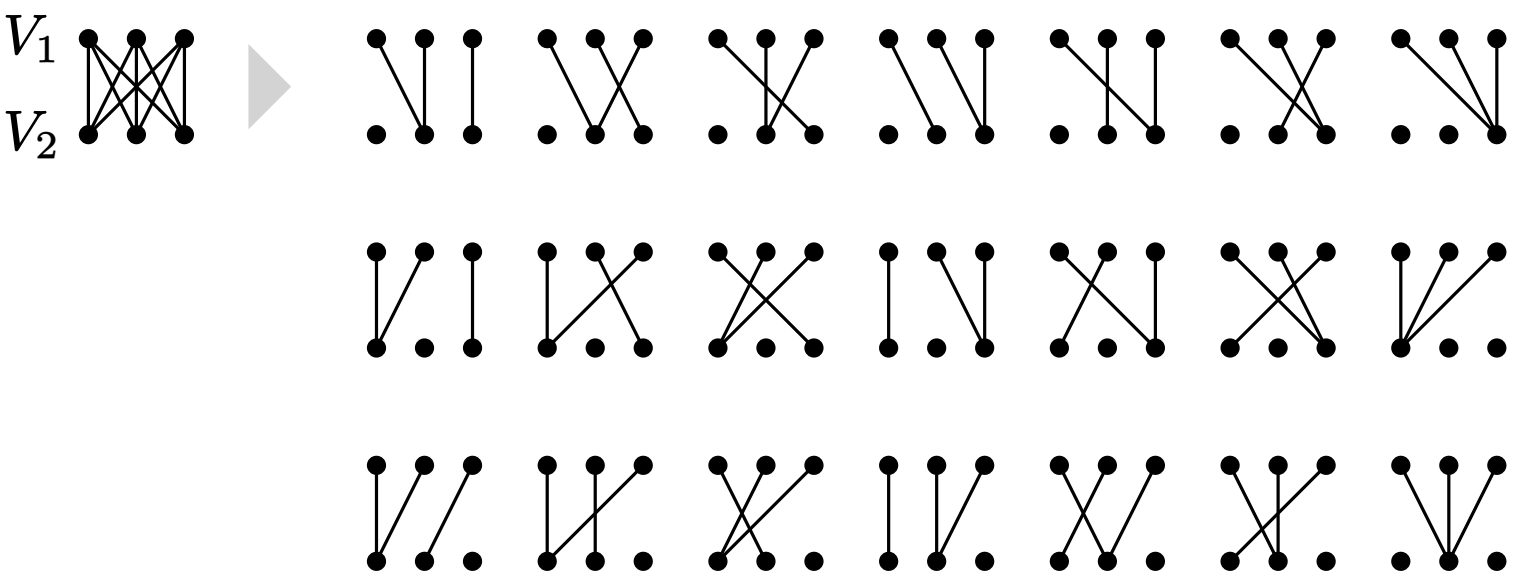
包除原理によるアルゴリズムの考え方

1. U と A_i を上手に定める
2. $|A_S|$ の計算法を与える

包除原理：U を定める

ポイント：U では 数えすぎる ようにする

$$U = \left\{ M \subseteq E \;\middle|\; \begin{array}{l} \text{二部グラフ } (V_1, V_2, M) \text{ において} \\ \text{各 } v \in V_1 \text{ の次数が } 1 \end{array} \right\}$$

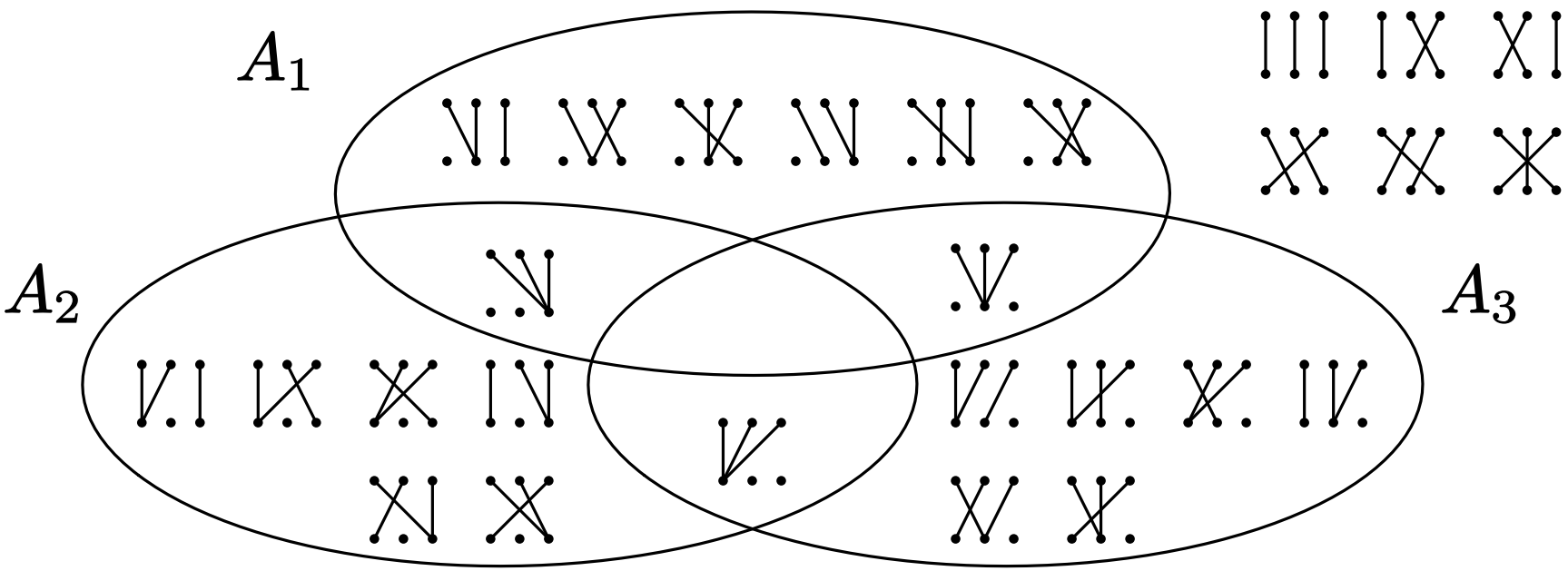


完全マッチング

包除原理：A_i を定める

$V_2 = \{1, 2, \ldots, n\}$ として，$i \in \{1, 2, \ldots, n\}$ に対して

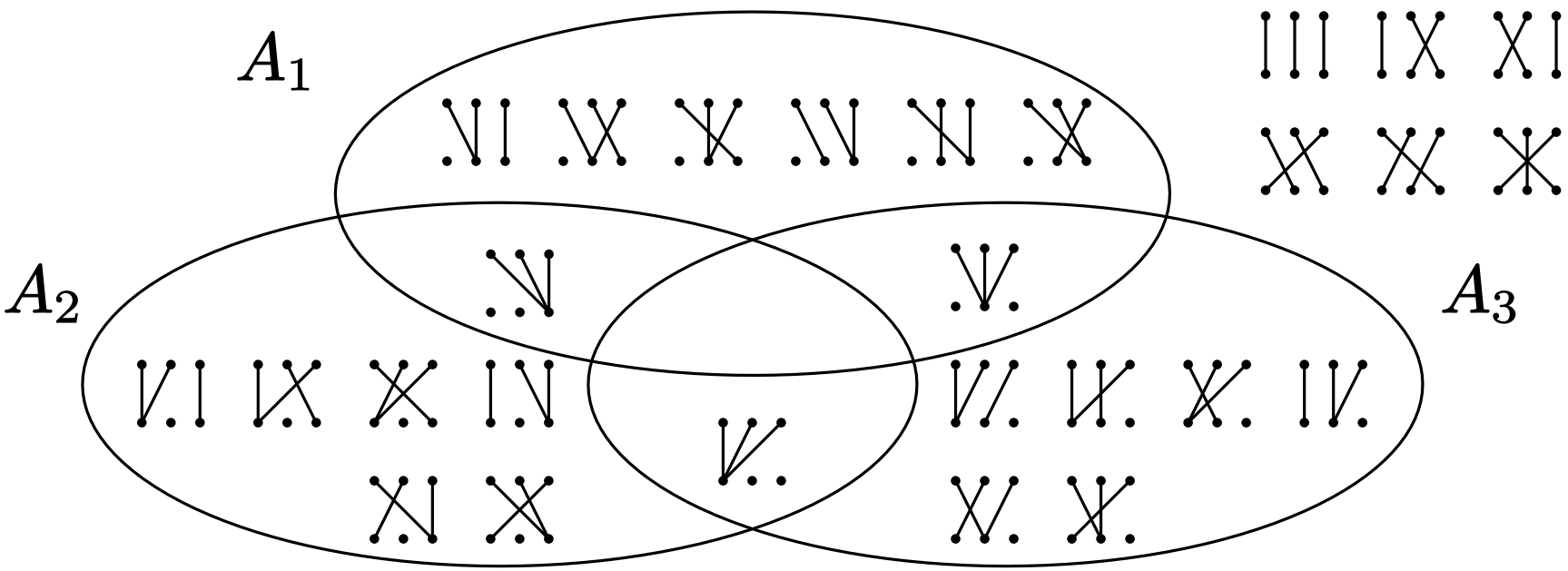
$$A_i = \left\{ M \subseteq E \;\middle|\; \begin{array}{l} \text{二部グラフ } (V_1, V_2, M) \text{ において，} \\ \text{各頂点 } v \in V_1 \text{ の次数は } 1 \text{ で} \\ \text{頂点 } i \in V_2 \text{ の次数は } 0 \end{array} \right\}$$



包除原理：A_i を定める

$V_2 = \{1, 2, \ldots, n\}$ として，$i \in \{1, 2, \ldots, n\}$ に対して

$$A_i = \left\{ M \subseteq E \;\middle|\; \begin{array}{l} \text{二部グラフ } (V_1, V_2, M) \text{ において，} \\ \text{各頂点 } v \in V_1 \text{ の次数は } 1 \text{ で} \\ \text{頂点 } i \in V_2 \text{ の次数は } 0 \end{array} \right\}$$



M が完全マッチング $\Leftrightarrow M \in \overline{A_1 \cup A_2 \cup \cdots \cup A_n}$

包除原理：$|A_S|$ の計算法

このとき，$S \subseteq \{1, 2, \ldots, n\}$ に対して

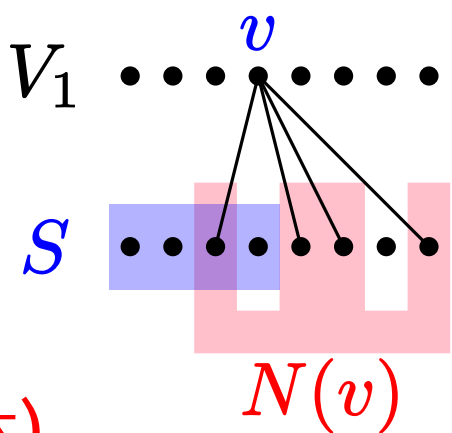
$$M \in A_S = \bigcap_{i \in S} A_i$$

$\Leftrightarrow$ 二部グラフ (V_1, V_2, M) において
各頂点 $v \in V_1$ の次数は 1，各頂点 $i \in S$ の次数は 0

すなわち，

$$|A_S| = \prod_{v \in V_1} |N(v) - S|$$

$N(v)$: v の開近傍
(v の隣接頂点全体)



(各 S に対して，$O^*(1)$ 時間で計算できる)

注：$S = \emptyset$ のときも，この式は正しい

包除原理：アルゴリズム

アルゴリズム ryser($G = (V_1, V_2, E)$) // $V_2 = [n]$

1. 各 $S \subseteq V_2$ に対して，次を計算

$$|A_S| = \prod_{v \in V_1} |N(v) - S|$$

2. 次を計算して，出力

$$\sum_{S \subseteq [n]} (-1)^{|S|} |A_S|$$

包除原理：計算量の導出

アルゴリズム ryser$(G = (V_1, V_2, E))$ // $V_2 = [n]$

1. 各 $S \subseteq V_2$ に対して，次を計算 $O^*(2^n)$

$$|A_S| = \prod_{v \in V_1} |N(v) - S|$$

2. 次を計算して，出力 $O^*(2^n)$

$$\sum_{S \subseteq [n]} (-1)^{|S|} |A_S|$$

包除原理：計算量の導出

アルゴリズム ryser$(G = (V_1, V_2, E))$ // $V_2 = [n]$

1. 各 $S \subseteq V_2$ に対して，次を計算 $O^*(2^n)$

$$|A_S| = \prod_{v \in V_1} |N(v) - S|$$

2. 次を計算して，出力 $O^*(2^n)$

$$\sum_{S \subseteq [n]} (-1)^{|S|} |A_S|$$

$\therefore$ 計算量 $= O^*(2^n)$

包除原理：計算量の導出

アルゴリズム ryser$(G = (V_1, V_2, E))$ // $V_2 = [n]$

1. 各 $S \subseteq V_2$ に対して，次を計算 $O^*(2^n)$

$$|A_S| = \prod_{v \in V_1} |N(v) - S|$$

2. 次を計算して，出力 $O^*(2^n)$

$$\sum_{S \subseteq [n]} (-1)^{|S|} |A_S|$$

$\therefore$ 計算量 $= O^*(2^n)$

注意：ビット長

包除原理：計算量の導出

アルゴリズム ryser$(G = (V_1, V_2, E))$ // $V_2 = [n]$

1. 各 $S \subseteq V_2$ に対して，次を計算 $O^*(2^n)$

$$|A_S| = \prod_{v \in V_1} |N(v) - S|$$

2. 次を計算して，出力 $O^*(2^n)$

$$\sum_{S \subseteq [n]} (-1)^{|S|} |A_S|$$

$\therefore$ 計算量 $= O^*(2^n)$　　メモリ使用量 $= O^*(2^n)$

注意：ビット長

包除原理：メモリ使用量の削減

アルゴリズム ryser-pspace($G = (V_1, V_2, E)$) // $V_2 = [n]$

1. sum $= 0$ // 初期化
2. 各 $S \subseteq V_2$ に対して，次を実行
   (a) term $= \prod_{v \in V_1} |N(v) - S|$ // term $= |A_S|$
   (b) sum $=$ sum $+ (-1)^{|S|}$ term
3. sum を出力

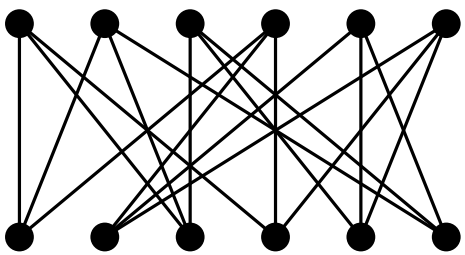
$\therefore$ 計算量 $= O^*(2^n)$ メモリ使用量 $= O^*(1)$

二部完全マッチングの数え上げ (再)

包除原理を使って，次を導いた

定理：Ryser の公式 (’63)

二部完全マッチングの数え上げは $O^*(2^n)$ 時間でできる
$(n = |V_1| = |V_2|)$



$\blacktriangleright$ 20

公式としては次の形になる

二部グラフ (V_1, V_2, E) の完全マッチングの総数 $= \sum_{S \subseteq V_2} (-1)^{|S|} \prod_{v \in V_1} |N(v) - S|$

二部完全マッチング：補足

二部完全マッチングの総数　　#P 完全 (難しい)

counting problem

二部完全マッチングの存在　　P (簡単)

decision problem　　(増加道法)

二部完全マッチング：補足

二部完全マッチングの総数 #P 完全 (難しい)

counting problem

二部完全マッチングの総数の偶奇 P (簡単)

parity problem (GF(2) における行列式計算)

二部完全マッチングの存在 P (簡単)

decision problem (増加道法)

観察：counting が解ける ⇒ parity が解ける

counting が解ける ⇒ decision が解ける

(解ける = 多項式時間で解ける)

本日の内容

1. 包除原理とは
2. 完全マッチングの数え上げ
3. **ハミルトン路の数え上げ**

- R.M. Karp, Dynamic programming meets the principle of inclusion and exclusion. *Operations Research Letters* 1 (1982) pp. 49–51.

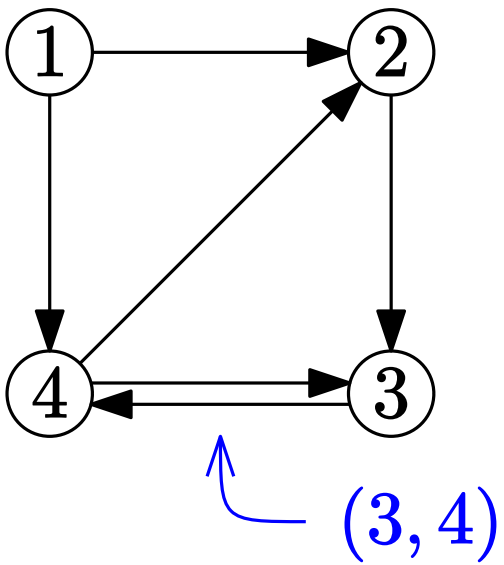
有向グラフの定義と用語

有向グラフ $G = (V, A)$ (directed graph)

頂点集合 (vertex set)　弧集合 (arc set)

V の要素 = 頂点

A の要素 = 弧



$$V = \{1, 2, 3, 4\}$$

$$A = \{(1, 2), (1, 4), (2, 3), (3, 4), (4, 2), (4, 3)\}$$

弧は (u, v) を省略して uv と書くことも

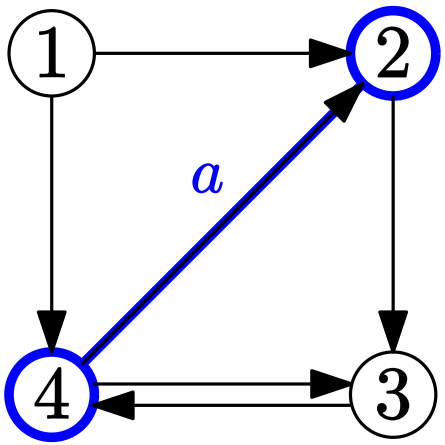
有向グラフの定義と用語

有向グラフ $G = (V, A)$ (directed graph)

頂点集合 (vertex set)　弧集合 (arc set)

V の要素 = 頂点

A の要素 = 弧



$V = \{1, 2, 3, 4\}$

$A = \{(1, 2), (1, 4), (2, 3), (3, 4), (4, 2), (4, 3)\}$

弧は (u, v) を省略して uv と書くことも

弧 $a = (u, v)$ を考えると

u は a の **始点**

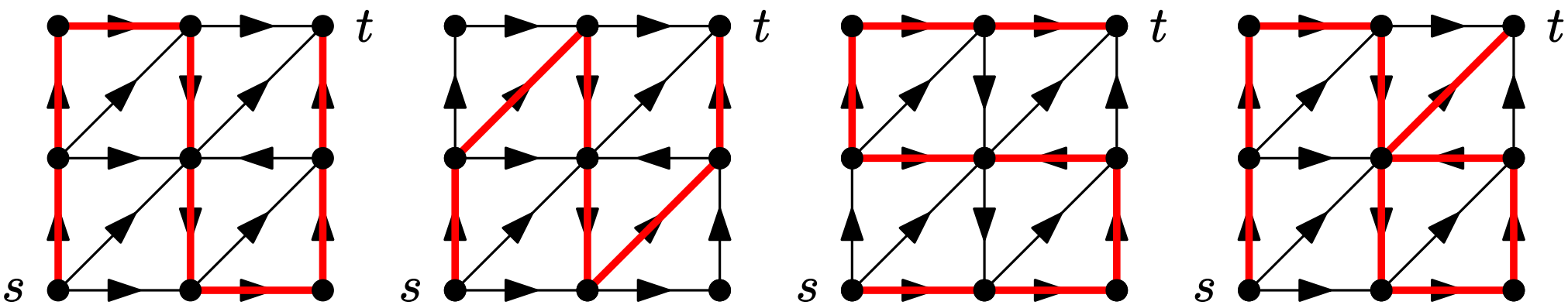
v は a の **終点**

ハミルトン路 (ハミルトン道)

有向グラフ $G = (V, A)$，2 頂点 $s, t \in V$

定義：ハミルトン路 (Hamiltonian path)

G において s から t へ至る **ハミルトン路** とは，
s を出発し，G のすべての頂点をちょうど一度ずつ通り，
t に到着する経路



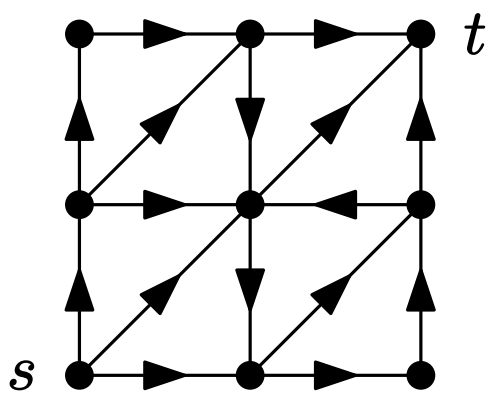
s から t へ至る
ハミルトン路
である

s から t へ至る
ハミルトン路
ではない

ハミルトン路の数え上げ

問題：ハミルトン路の数え上げ

入力： 有向グラフ $G = (V, A)$，2 頂点 $s, t \in V$
出力： G において s から t へ至るハミルトン路の総数



▶ 1

事実：ハミルトン路の数え上げは #P 完全

(Liskiewicz, Ogihara, Toda '03)

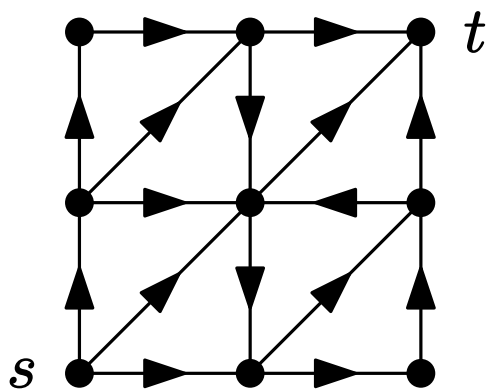
(注) #P 完全 ⇒ NP 困難

包除原理による数え上げ

包除原理を使って，次を導く

定理：ハミルトン路に対する包除原理 (Karp '82)

ハミルトン路の数え上げは $O^*(2^n)$ 時間でできる $(n = |V|)$

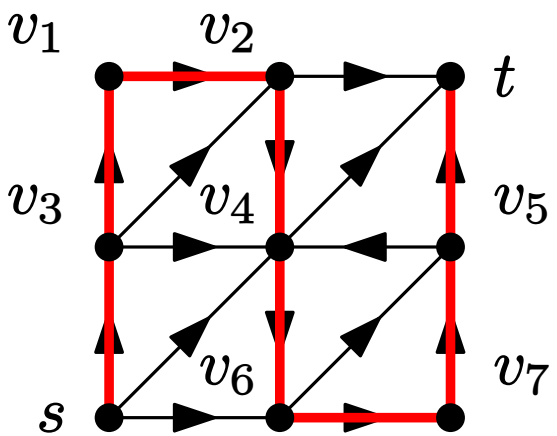


▶ 1

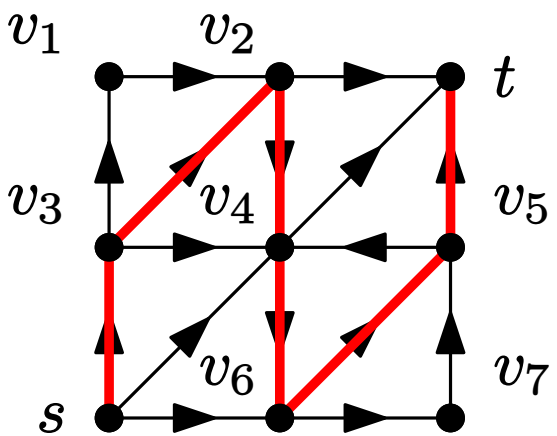
包除原理によるアルゴリズムの考え方

1. U と A_i を上手に定める
2. $|A_S|$ の計算法を与える

準備：路を頂点列とみなす

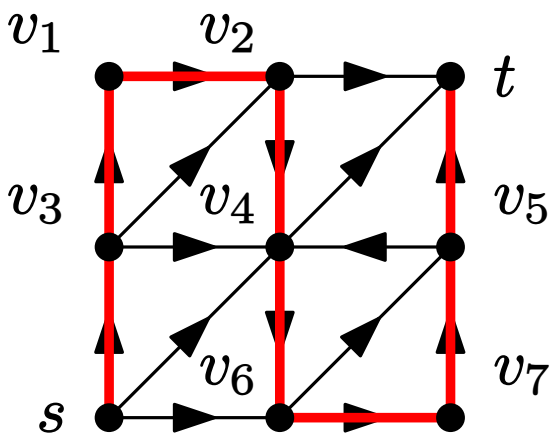


$\langle s, v_3, v_1, v_2, v_4, v_6, v_7, v_5, t \rangle$

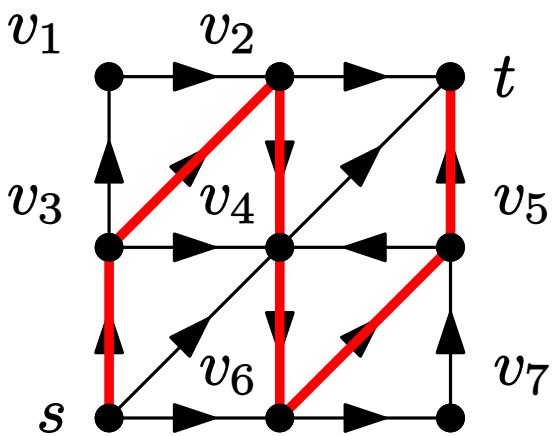


$\langle s, v_3, v_2, v_4, v_6, v_5, t \rangle$

準備：路を頂点列とみなす



$\langle s, v_3, v_1, v_2, v_4, v_6, v_7, v_5, t \rangle$



$\langle s, v_3, v_2, v_4, v_6, v_5, t \rangle$

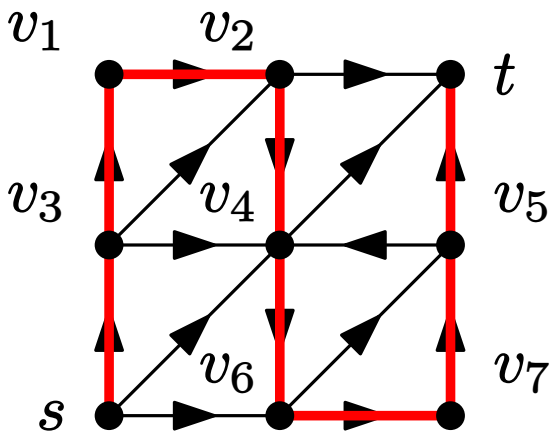
頂点列が s から t へ至るハミルトン路を表す $\Leftrightarrow$

- s で始まり，t で終わり，各頂点がちょうど一度現れる
- 長さ $=$ 頂点数
- 連続する 2 つの頂点 uv は弧を成す

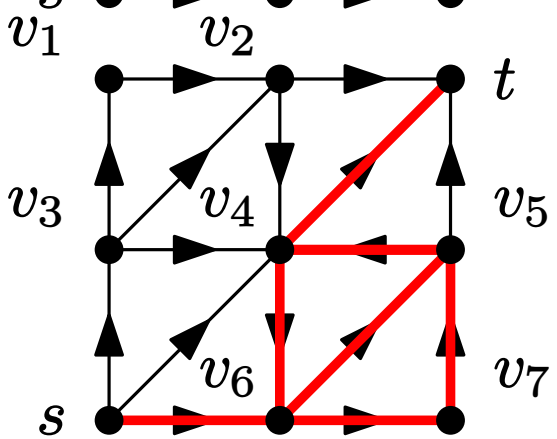
包除原理：U を定める

ポイント：U では 数えすぎる ようにする

頂点列が U の要素である $\Leftrightarrow$
- s で始まり，t で終わる
- 長さ $=$ 頂点数
- 連続する 2 つの頂点 uv は弧を成す



$\langle s, v_3, v_1, v_2, v_4, v_6, v_7, v_5, t \rangle$



$\langle s, v_6, v_7, v_5, v_4, v_6, v_5, v_4, t \rangle$

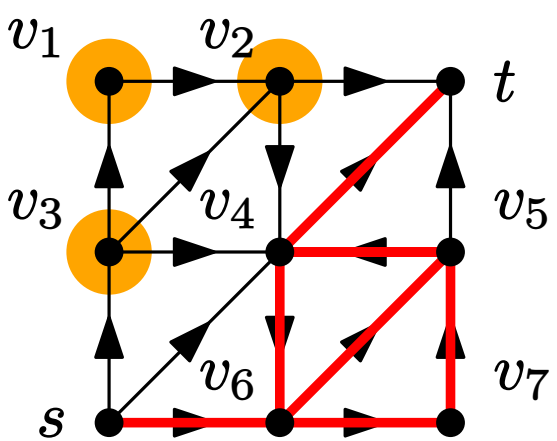
包除原理：A_i を定める

$V = \{v_1, v_2, \ldots, v_{n-2}, s, t\}$, $i \in \{1, 2, \ldots, n-2\}$ とする

頂点列 $\pi \in U$ に対して

$\pi \in A_i \quad \Leftrightarrow \quad \pi$ は v_i を通らない (含まない)

注：他の頂点は通っても通らなくてもよい



$$\langle s, v_6, v_7, v_5, v_4, v_6, v_5, v_4, t \rangle \in A_1 \cap A_2 \cap A_3 = A_{\{1,2,3\}}$$

このとき

π がハミルトン路に対応 $\quad \Leftrightarrow \quad \pi \in \overline{A_1 \cup A_2 \cup \cdots \cup A_{n-2}}$

包除原理：$|A_S|$ の計算法 (1)

動的計画法！

包除原理：$|A_S|$ の計算法 (1)

動的計画法！

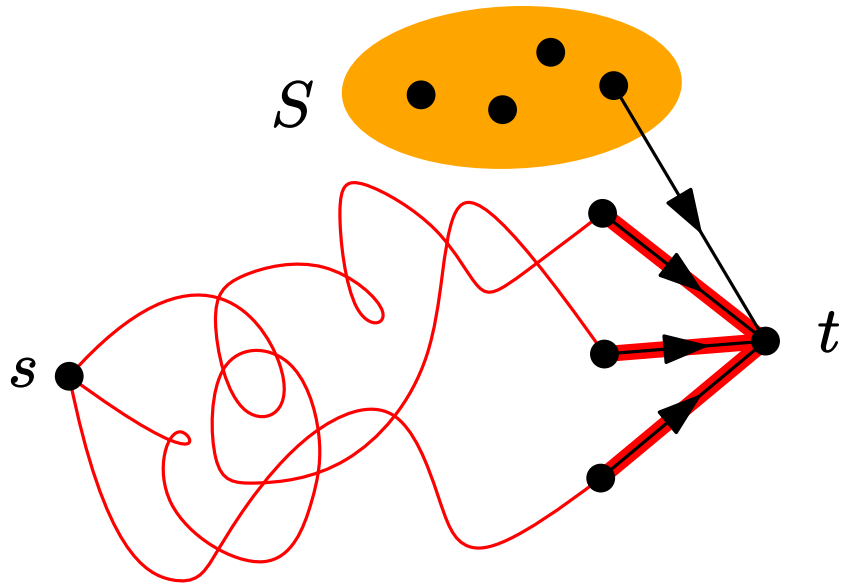
数え上げ対象

動的計画法を考えるときの鍵

1. ~~最適解~~の持つ **再帰的な構造** を見出す
2. 上の構造から **状態** を適切に定義する
3. 状態の間の **再帰式** を立てる

包除原理：$|A_S|$ の計算法 (1)

動的計画法！



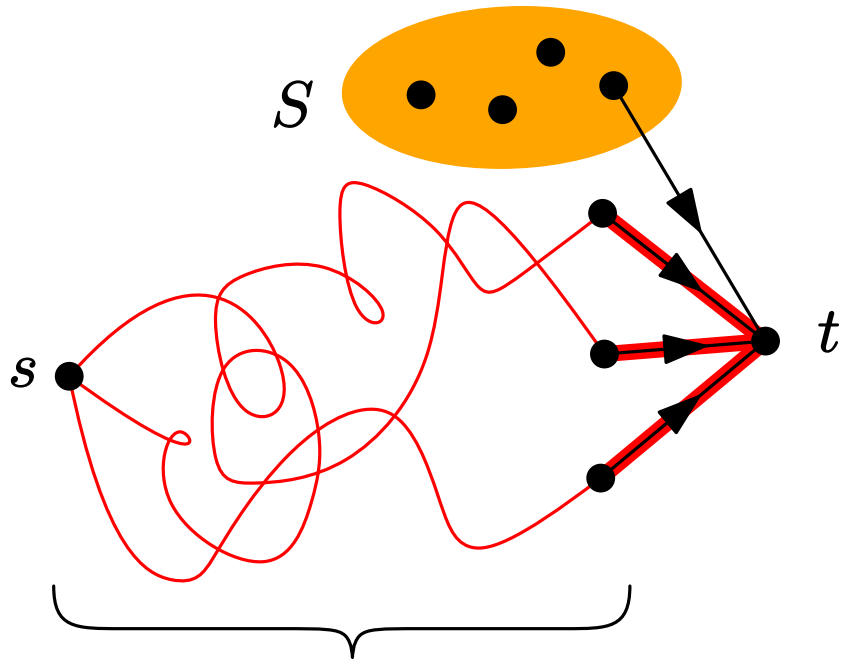
数え上げ対象

動的計画法を考えるときの鍵

1. ~~最適解~~の持つ **再帰的な構造** を見出す
2. 上の構造から **状態** を適切に定義する
3. 状態の間の **再帰式** を立てる

包除原理：$|A_S|$ の計算法 (1)

動的計画法！



数え上げ対象

動的計画法を考えるときの鍵

1. ~~最適解~~の持つ **再帰的な構造** を見出す
2. 上の構造から **状態** を適切に定義する
3. 状態の間の **再帰式** を立てる

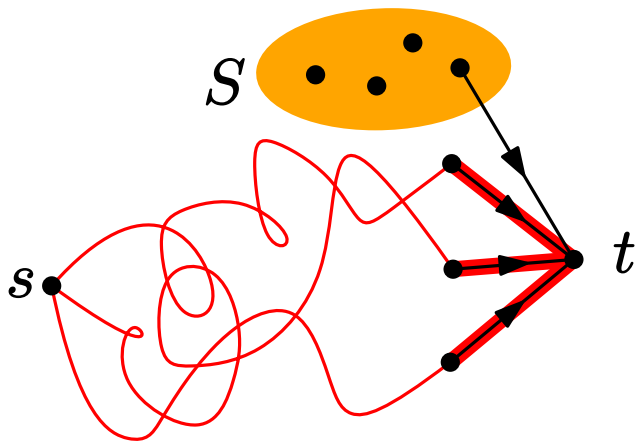
包除原理：$|A_S|$ の計算法 (2)

状態：(ℓ, v)　ただし，$\ell \in \{1, 2, \ldots, n\}, v \in V - S$

状態の値：$f(\ell, v) =$ 次を満たす頂点列 π の総数

- s で始まり，v で終わり，S の頂点を含まない
- 長さ $= \ell$
- 連続する 2 つの頂点 uv は弧を成す

最終的に出力する値：$f(n, t)$



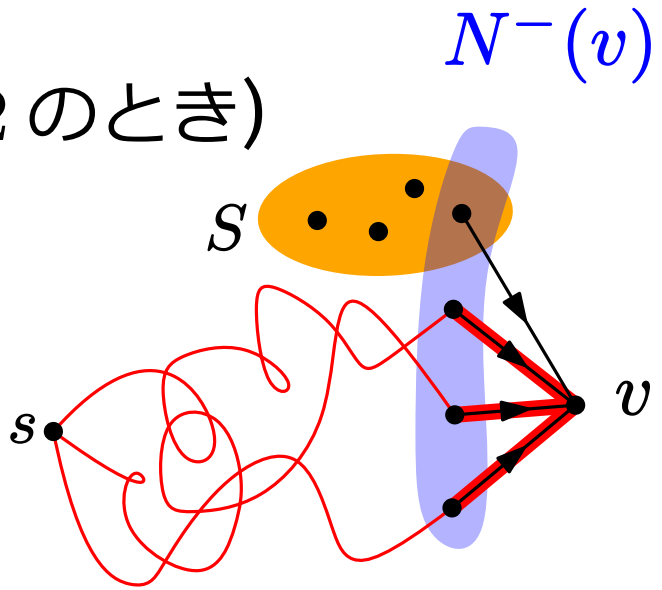
包除原理：$|A_S|$ の計算法 (3)

再帰式 (Bellman 方程式)：

$$f(1,v) = \begin{cases} 1 & (v = s) \\ 0 & (v \neq s) \end{cases}$$

$$f(\ell,v) = \sum_{u \in N^-(v) - S} f(\ell-1,u) \quad (\ell \geq 2 \text{ のとき})$$

v の開入近傍 (open in-neighborhood)



状態の値：$f(\ell,v)$ = 次を満たす頂点列 π の総数

- s で始まり，v で終わり，S の頂点を含まない
- 長さ $= \ell$
- 連続する 2 つの頂点 uv は弧を成す

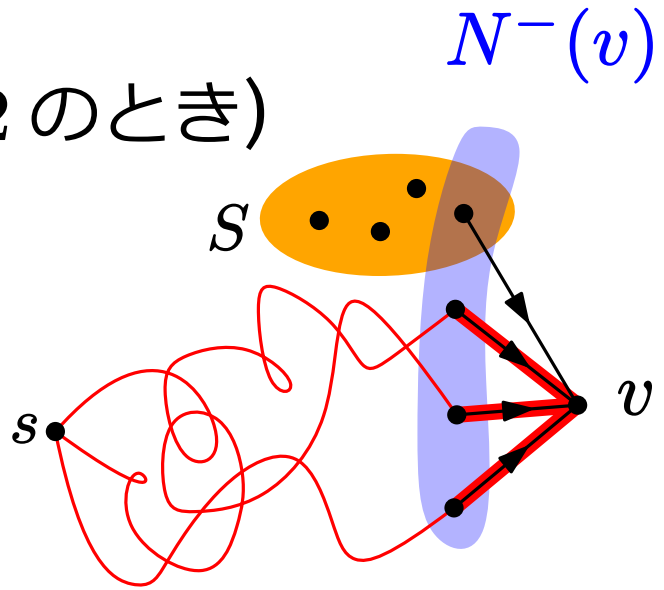
包除原理：$|A_S|$ の計算法 (3)

再帰式 (Bellman 方程式)：(注：$S = \emptyset$ のときも正しい)

$$f(1, v) = \begin{cases} 1 & (v = s) \\ 0 & (v \neq s) \end{cases}$$

$$f(\ell, v) = \sum_{u \in N^-(v) - S} f(\ell - 1, u) \quad (\ell \geq 2 \text{ のとき})$$

v の開入近傍 (かいいりきんぼう) (open in-neighborhood)



状態の値：$f(\ell, v) =$ 次を満たす頂点列 π の総数

- s で始まり，v で終わり，S の頂点を含まない
- 長さ $= \ell$
- 連続する 2 つの頂点 uv は弧を成す

包除原理：$|A_S|$ の計算法 (4) まとめ

アルゴリズム walks-dp$(G=(V,A),s,t,S)$ // $n=|V|$

1. $f(1,s)=1$; $f(1,v)=0\ \forall\, v\neq s$
2. すべての $\ell\in\{2,\ldots,n\}$ と $v\in V-S$ に対して，
   ℓ の小さい方から順に $f(\ell,v)$ を再帰式に基づいて計算
3. $f(n,t)$ を出力

再帰式 (Bellman 方程式)：$f(\ell,v)=\displaystyle\sum_{u\in N^-(v)-S} f(\ell-1,u)$

包除原理：$|A_S|$ の計算法 (4) まとめ

アルゴリズム walks-dp$(G=(V,A),s,t,S)$ // $n=|V|$

1. $f(1,s)=1$; $f(1,v)=0\ \forall\, v\neq s$
2. すべての $\ell\in\{2,\ldots,n\}$ と $v\in V-S$ に対して，
   ℓ の小さい方から順に $f(\ell,v)$ を再帰式に基づいて計算
3. $f(n,t)$ を出力

再帰式 (Bellman 方程式)：$f(\ell,v)=\sum_{u\in N^-(v)-S} f(\ell-1,u)$

$\therefore$ 各 S に対して，計算量 $=O^*(1)$，メモリ使用量 $=O^*(1)$

包除原理：アルゴリズム

アルゴリズム hampaths$(G = (V, A), s, t)$

1. sum $= 0$　　// 初期化
2. 各 $S \subseteq V - \{s, t\}$ に対して，次を実行
   (a) term $=$ walks-dp(G, s, t, S)　　// term $= |A_S|$
   (b) sum $=$ sum $+ (-1)^{|S|}$ term
3. sum を出力

包除原理：アルゴリズム

アルゴリズム hampaths($G = (V, A), s, t$)

1. sum $= 0$　　// 初期化
2. 各 $S \subseteq V - \{s, t\}$ に対して，次を実行　$O^*(2^n)$回の繰返し
   (a) term $=$ walks-dp(G, s, t, S)　　// term $= |A_S|$
   (b) sum $=$ sum $+ (-1)^{|S|}$ term
   ］ $O^*(1)$
3. sum を出力

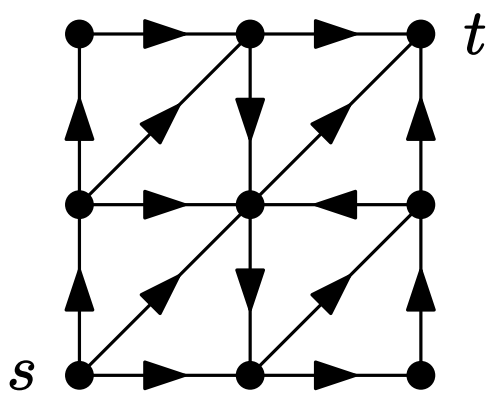
$\therefore$ 計算量 $= O^*(2^n)$　　メモリ使用量 $= O^*(1)$

ハミルトン路の数え上げ (再)

包除原理を使って，次を導いた

定理：ハミルトン路に対する包除原理 (Karp '82)

ハミルトン路の数え上げは $O^*(2^n)$ 時間でできる ($n = |V|$)



$\blacktriangleright$ 1

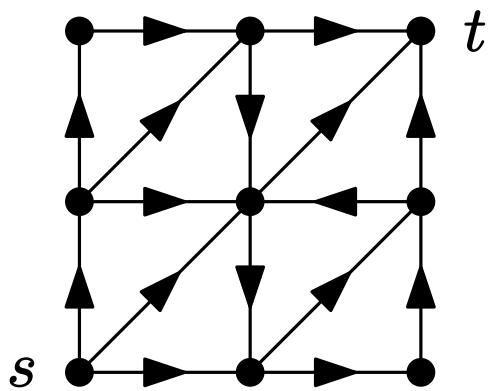
事実：ハミルトン路の数え上げは #P 完全

(Liskiewicz, Ogihara, Toda '03)

(注) #P 完全 $\Rightarrow$ NP 困難

ハミルトン路の見つけ方

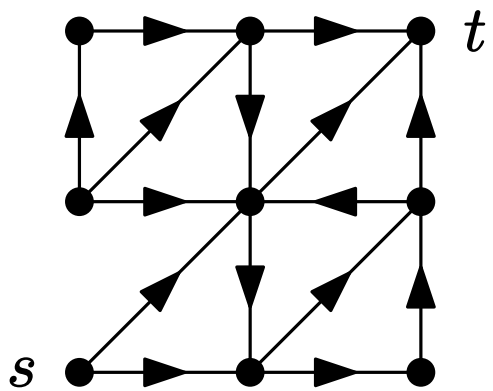
「数え上げ」から「発見」へ



1

ハミルトン路の見つけ方

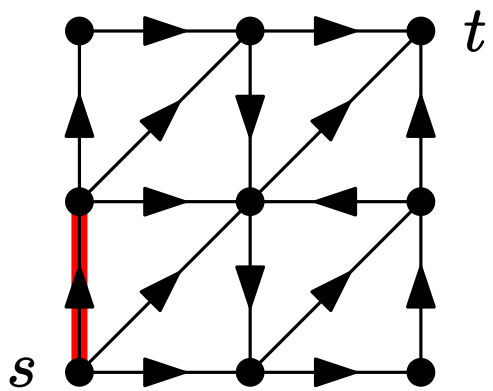
「数え上げ」から「発見」へ



0

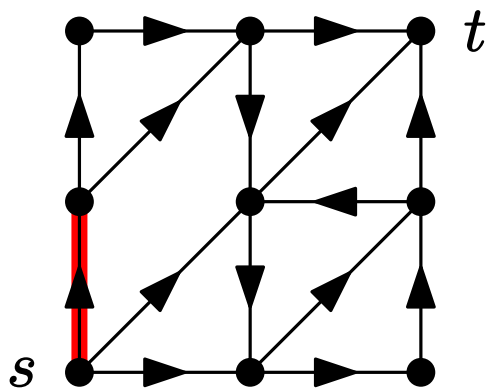
ハミルトン路の見つけ方

「数え上げ」から「発見」へ



ハミルトン路の見つけ方

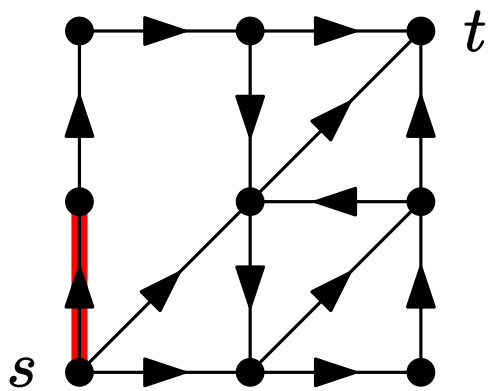
「数え上げ」から「発見」へ



1

ハミルトン路の見つけ方

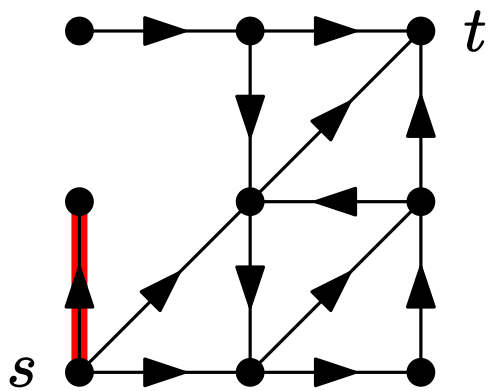
「数え上げ」から「発見」へ



1

ハミルトン路の見つけ方

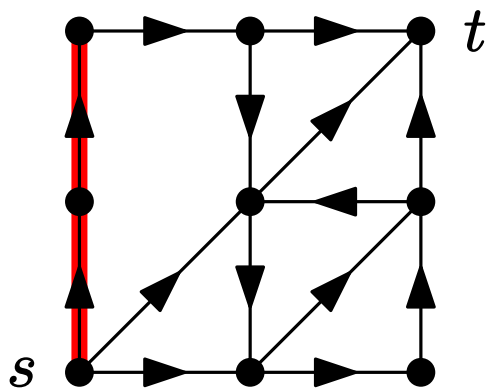
「数え上げ」から「発見」へ



0

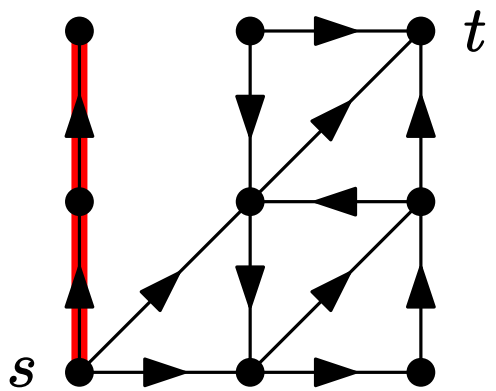
ハミルトン路の見つけ方

「数え上げ」から「発見」へ



ハミルトン路の見つけ方

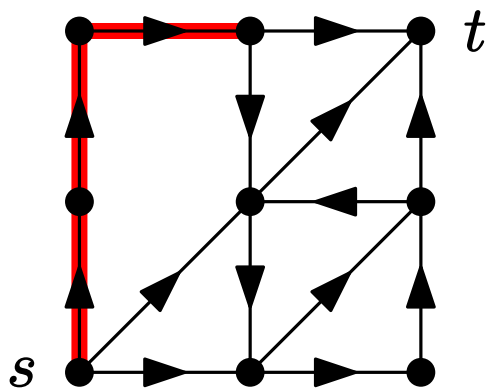
「数え上げ」から「発見」へ



0

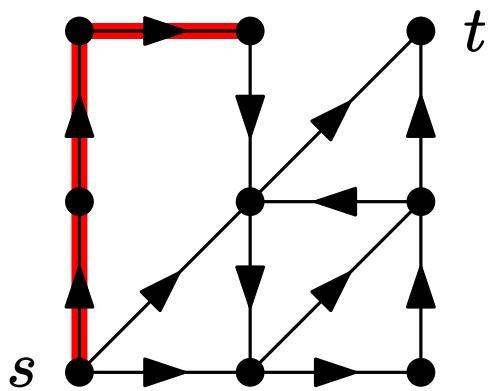
ハミルトン路の見つけ方

「数え上げ」から「発見」へ



ハミルトン路の見つけ方

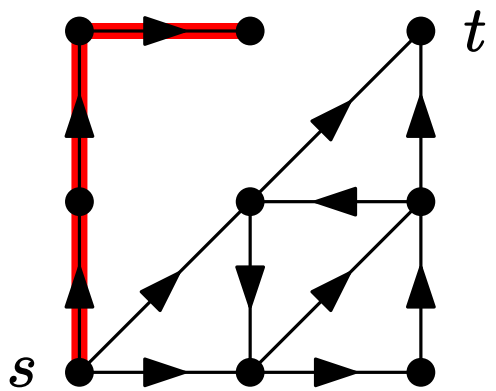
「数え上げ」から「発見」へ



1

ハミルトン路の見つけ方

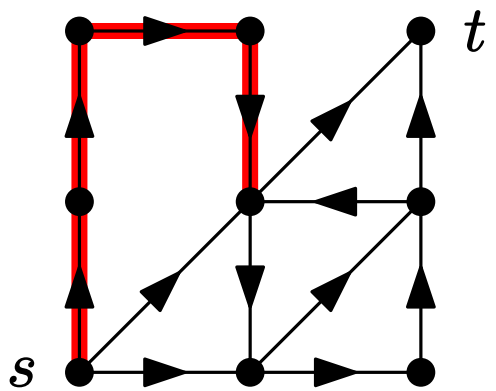
「数え上げ」から「発見」へ



0

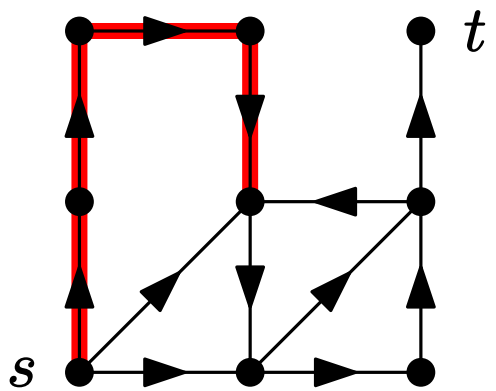
ハミルトン路の見つけ方

「数え上げ」から「発見」へ



ハミルトン路の見つけ方

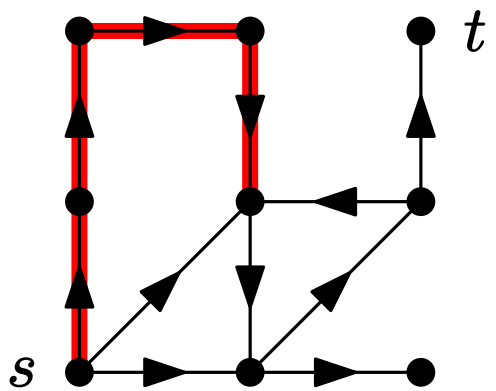
「数え上げ」から「発見」へ



1

ハミルトン路の見つけ方

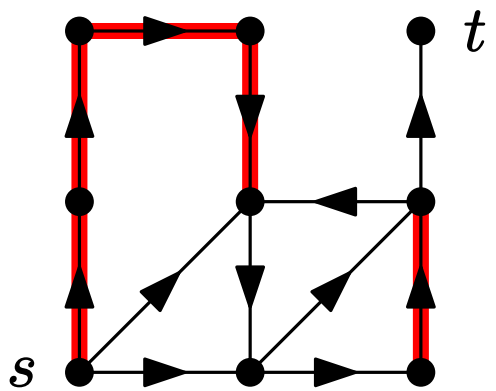
「数え上げ」から「発見」へ



0

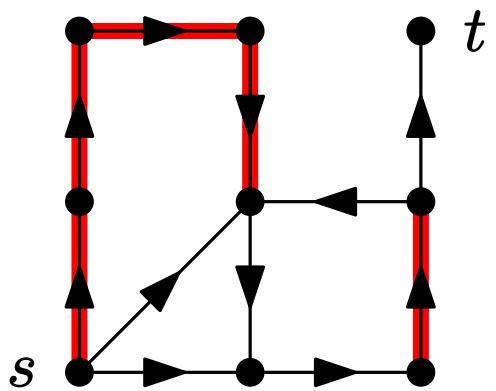
ハミルトン路の見つけ方

「数え上げ」から「発見」へ



ハミルトン路の見つけ方

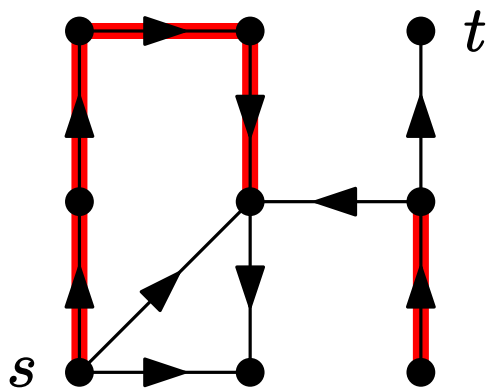
「数え上げ」から「発見」へ



1

ハミルトン路の見つけ方

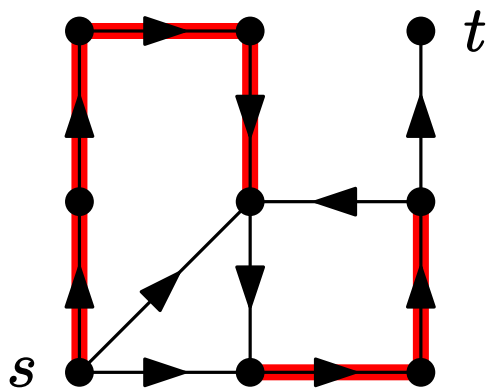
「数え上げ」から「発見」へ



0

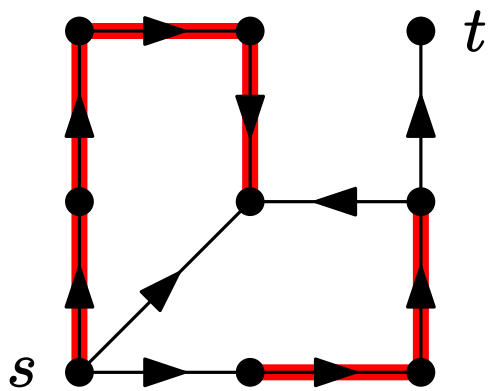
ハミルトン路の見つけ方

「数え上げ」から「発見」へ



ハミルトン路の見つけ方

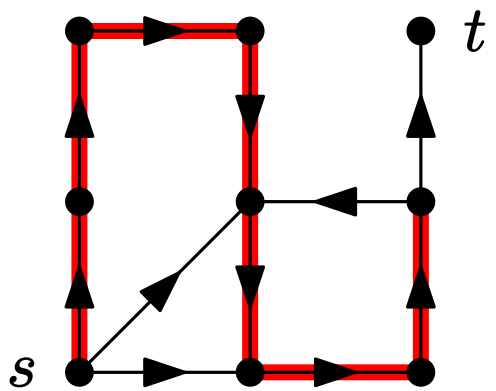
「数え上げ」から「発見」へ



0

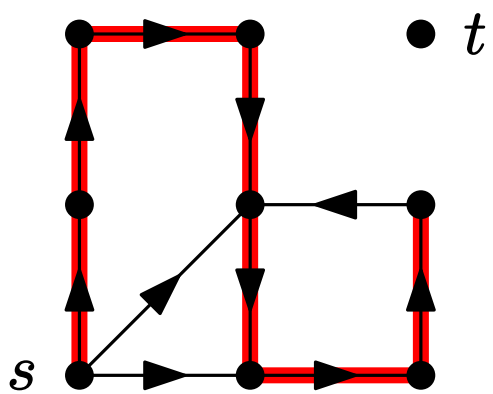
ハミルトン路の見つけ方

「数え上げ」から「発見」へ



ハミルトン路の見つけ方

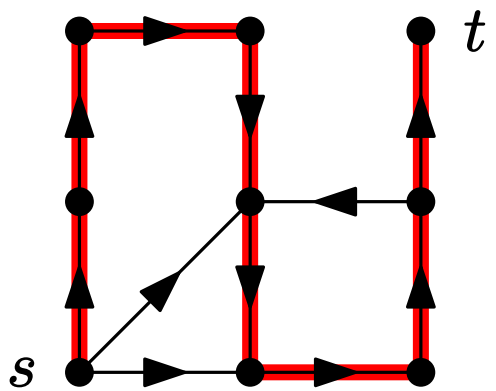
「数え上げ」から「発見」へ



0

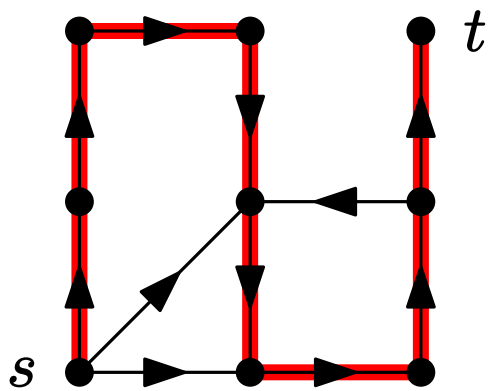
ハミルトン路の見つけ方

「数え上げ」から「発見」へ



ハミルトン路の見つけ方

「数え上げ」から「発見」へ



帰結

ハミルトン路が存在すれば,
その 1 つを $O^*(2^n)$ 時間で見つけられる

補足：ハミルトン路の数え上げ

ハミルトン路の総数	#P 完全 (難しい)
counting problem	(Liskiewicz, Ogihara, Toda '03)
ハミルトン路の総数の偶奇	⊕P (パリティピー) 完全 (難しい)
parity problem	(Valiant '05)
ハミルトン路の存在	NP 完全 (難しい)
decision problem	(Karp '72)

観察：counting が解ける ⇒ parity が解ける

counting が解ける ⇒ decision が解ける

(解ける = 多項式時間で解ける)

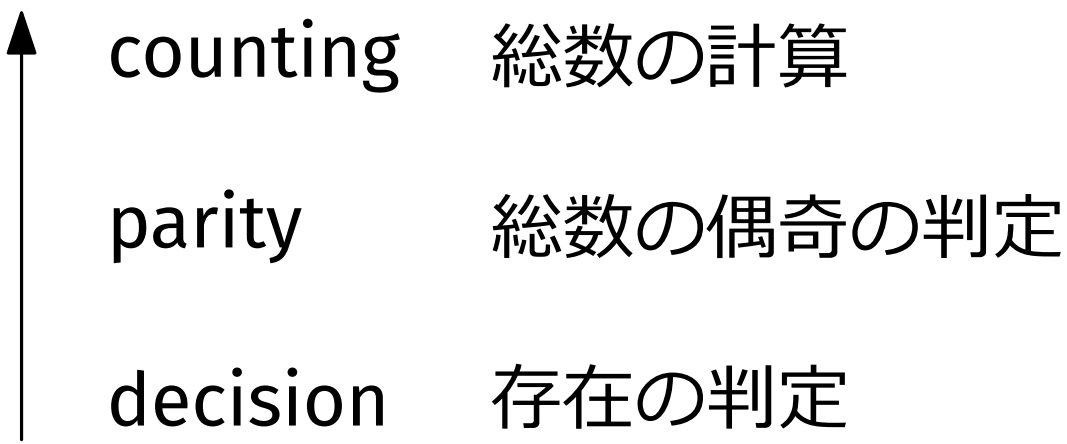
補足の補足：偶奇と判定の関係

疑問：parity と decision の関係は？

Valiant-Vazirani の定理 (’86) の帰結

ハミルトン路の偶奇が多項式時間で計算できる ⇒
ハミルトン路の存在判定が (乱択) 多項式時間でできる

気分



この辺りは計算複雑性理論のだんだん深いところに踏み込んでいる

まとめと次回の予告

今回と次回

包除原理 (inclusion-exclusion principle) による
アルゴリズムの設計と解析

今回

- 包除原理の説明
  - 二部完全マッチングの数え上げ
  - ハミルトン路の数え上げ

次回

- 包除原理による彩色問題の解法 ($O^*(2^n)$ 時間)