I216 Computational Complexity and Discrete Mathematics

by

Prof. Ryuhei Uehara

and

Prof. Atsuko Miyaji

I216 <u>計算量の理論</u>と離散数学

<u>上原隆平</u>、宮地充子

Computational Complexity

- Goal 1:
  - "*Computable Function/Problem/Language/Set*"

- Goal 2:
  - How can you show "*Difficulty of Problem*"
    - There are *intractable* problems even if they are computable!
      - because they require too many resources (time/space)!
    - Technical terms;

      The class NP, P≠NP conjecture, NP-hardness, reduction

計算量の理論

- ゴール1:
  - “計算可能な*関数/問題/言語/集合*”

- ゴール2:
  - 「問題の困難さ」を示す方法を学ぶ
    - 計算可能な問題であっても、*手におえない* 場合がある！
      - 計算に必要な資源（時間・領域）が多すぎるとき
    - 関連する専門用語;

      クラスNP, P≠NP予想, NP困難性, 還元

5. Computational Complexity

5.0. Overview

- "Computable?"

➔How much cost is required for computation?

  – Computational Complexity Theory

  1. Studies on <u>upper bound</u> of computational cost
  2. Studies on <u>lower bound</u> of computational cost
  3. Structural studies on hardness of computation

5. 計算量の理論

5.0. 概要

- 「計算可能」

➔計算にどのくらいの資源（コスト）が必要か？

  – 計算量の理論

1. 計算量の<u>上界</u>の研究
2. 計算量の<u>下界</u>の研究
3. 計算の困難性の階層構造の研究

5. Computational Complexity

5.0.1. **Studies on upper bound of computational cost**

Algorithm Theory: design of efficient algorithms

- Suppose we have an algorithm A which solves a problem X in at most time $T(n)$ for any input of size n.
- Then, an upper bound on the time complexity of the algorithm A is $T(n)$.

  [asymptotic worst case time complexity]

5.計算量の理論

5.0.1. 計算量の上界の研究

アルゴリズム論: 効率の良いアルゴリズムの設計

- アルゴリズム A が問題 X を大きさ n のどんな入力に対しても高々 $T(n)$ 時間で解けるとする.
- このとき, アルゴリズム A の時間計算量の上界は $T(n)$ である.

  [最悪な場合の漸近的な時間計算量]

5. Computational Complexity

5.0.2. Studies on lower bound of computational cost

If <u>*any*</u> algorithm for a problem X takes time $T(n)$ in the worst case, a lower bound on the time complexity of the problem X is $T(n)$.

- $\mathcal{P} \neq \mathcal{NP}$ conjecture
- Robustness of crypto system

5.0.3. Structural studies on hardness of computation

Studies to characterize hardness in the level of "*xx*-hardness" hierarchical structure depending on the hardness

5.計算量の理論

5.0.2. 計算量の下界の研究

問題Xに対する<u>どんな</u>アルゴリズムも最悪の場合 $T(n)$ 時間かかるとき，問題Xの時間計算量の下界は $T(n)$ である.

- $\mathcal{P} \neq \mathcal{NP}$ 予想
- 暗号システムの頑健性

5.0.3. 計算の困難性の階層構造の研究

問題の困難性に関する概念「xx困難性」の階層構造の特徴付けを研究する

5. Computational Complexity

5.1. Measuring Computation Time

5.1.1. Time complexity of a Turing machine

Definition:

Let *M* be a (deterministic) Turing machine that halts on all inputs.

The *running time* or *time complexity* of *M* is the function $f: \mathcal{N} \to \mathcal{N}$ s.t.

f(n) is the maximum number of steps of *M(x)* for all *x* of length *n*

In the case,

- we say that *M* runs in time *f(n)*
- *M* is an *f(n)* time Turing machine

We need further *tools* to estimate/compare algorithms

[Note]

- *f(n)* takes the maximum for all strings of length *n* (worst case complexity)
- usually, *f(n)* may be monotone increasing function, but…?

5.計算量の理論

5.1. 計算時間の評価

5.1.1. チューリングマシンの時間計算量

定義: どんな入力に対しても停止する(決定性)チューリングマシンを*M*とする．このとき*M*の**<u>実行時間</u>**または**<u>時間計算量</u>**は以下を満たす関数 $f:\mathcal{N}\to\mathcal{N}$ sである：.

f(n) は長さ*n*のすべての入力*x*に対する*M*(*x*)のステップ数の最大値

このとき，
- *M*の<u>実行時間</u>は *f(n)*時間
- *M*は<u>*f(n)*時間チューリングマシン</u>という

アルゴリズムを評価/比較するためにはさらなる道具が必要.

[注意]
- *f(n)* は長さ*n*の文字列すべてに対する<u>最大値</u>をとっている (最悪の場合の実行時間)
- 通常，*f(n)* は単調増加関数だが…?

5. Computational Complexity

5.1. Measuring Computation Time

5.1.2. Big-O notation

Definition: For functions f and g on natural numbers, if

$\exists c, n_0 > 0,\ \forall n \geqq n_0\ [f(n) \leqq c\, g(n)]$

then we say $f(n)$ is in the order of $g(n)$ and denote it by $f(n) = O(g(n))$.

Remark: the constants c and n_0 must be determined independently of n.

Ex. 1: The followings hold for any functions f, g and h on natural numbers:

1. $\forall n [f(n) \leqq g(n)] \rightarrow f(n) = O(g(n))$
2. $[f(n) = O(g(n))$ and $g(n) = O(h(n))] \rightarrow f(n) = O(h(n))$

Ex. 2: Prove the following:

1. $5n^3+4n^2+n=O(n^3)$
2. $5n^3+4n^2+n=O(n^4)$
3. $5n^3+4n^2+n \neq O(n^2)$

[Comment] Some people write as $f(n) \in O(g(n))$

5.計算量の理論

5.1.計算時間の評価

5.1.2. O記法

定義: 自然数上の関数 f と g に対し,
$\exists c, n_0 > 0, \forall n \geqq n_0 [f(n) \leqq c\, g(n)]$
が成立するなら, $f(n)$ は <u>オーダー $g(n)$</u> の関数といい, $f(n) = O(g(n))$ と書く.

<u>注意</u>: 定数 c と n_0 はn とは<u>独立に</u>決められる必要がある.

例1: 自然数上の任意の関数 f, g, h に対して以下が成立:
1. $\forall n[f(n) \leqq g(n)] \rightarrow f(n) = O(g(n))$
2. $[f(n) = O(g(n)) \text{ and } g(n) = O(h(n))] \rightarrow f(n) = O(h(n))$

例2: 以下を示せ:
1. $5n^3+4n^2+n=O(n^3)$
2. $5n^3+4n^2+n=O(n^4)$
3. $5n^3+4n^2+n \neq O(n^2)$

[コメント] $f(n) \in O(g(n))$ と書く人もいる

5. Computational Complexity

5.1. Measuring Computation Time

5.1.3. Time complexity of a problem

Definition: Let Φ be a computing problem and $f(n)$ be a function over natural numbers. If we have a program A to compute Φ that runs in time $f(n)$ such that

$\exists c, n_0 > 0, \forall n \geqq n_0 \, [\, f(n) \leqq c\, g(n)]$

then we say that Φ is computable in $O(f(n))$ time, or time complexity of Φ is $O(f(n))$.

Remark: the constants c and n_0 must be determined independently of n.

Intuitively,

"problem Φ is computable within time $f(n)$" means

- time complexity of A may be less than $f(n)$.
- there may be a faster program to compute Φ than A does.

It only gives **an upper bound** of the complexity of the problem.

Our analysis may be improved

Better algorithm may be found

5.計算量の理論

5.1.計算時間の評価

5.1.3. 問題の時間計算量

定義: Φ を計算問題, $f(n)$ を自然数上の関数とする.
Φ を計算するプログラムA が $f(n)$ 時間で動作して以下を満たすとする.
$\exists c, n_0>0, \forall n \geqq n_0 \, [\, f(n) \leqq c\, g(n)]$
このとき Φ は $O(f(n))$ 時間で計算可能, または Φの時間計算量は$O(f(n))$である.

注意: 定数 c と n_0 はn とは独立に決められる必要がある.

直観的には,
「問題 Φ が $f(n)$時間で計算可能」というとき,
・A の時間計算量は$f(n)$より小さいかもしれない
・A よりも速くΦ を計算するアルゴリズムがあるかもしれない
問題の複雑さに関する上界を与えているにすぎない

解析を改善できるかもしれない

より速いアルゴリズムが
見つかるかもしれない

5.*. A history of the PRIME problem

PRIME
Input : a natural number *n* (binary representation)
Question: Is *n* prime?
PRIME $\equiv$ {*n* : *n* is prime}={2,3,5,7,11,13,17,…}

Stirling's Formula :

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

```
program Naive(input n);
begin
  for each i:= 1 < i < n do
    if n mod i = 0 then reject end-if
  end-for;
  accept
end.
```

try to divide by numbers between 2～*n*-1

log *n*･log *i* time

$O(l^6)$ time algorithm was developed in 2002!!

running time $\leq \sum_{1<i<n}(c \log n \log i + d)$

$$= c\log n \log n! + dn = O(n(\log n)^2)$$

When the length of *n* is *l*, *l* is approximately log *n*. So, the running time is $O(l^2 2^l)$.
Thus, time complexity of PRIME is $O(l^2 2^l)$.

5.*. 素数判定問題の歴史

PRIME
入力 : 自然数 n (2進数で表現)
質問: n は素数か？
PRIME $\equiv\{n : n$ は素数$\}=\{2,3,5,7,11,13,17,\ldots\}$

スターリングの公式 :

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

```
program Naive(input n);
begin
  for each i:= 1 < i < n do
    if n mod i = 0 then reject end-if
  end-for;
  accept
end.
```

2〜n-1までのすべての数で実際に割ってみる

log n・log i time

$O(l^6)$時間アルゴリズムが2002年に開発された!!

$$\text{実行時間} \le \sum_{1<i<n}(c \log n \log i + d)$$
$$= c \log n \log n! + dn = O(n(\log n)^2)$$

自然数 n を表現した文字列の長さを l とすると, l はおおよそ log nである.
したがって実行時間は $O(l^2 2^l)$ である.
よって PRIME の時間計算量は $O(l^2 2^l)$.である.

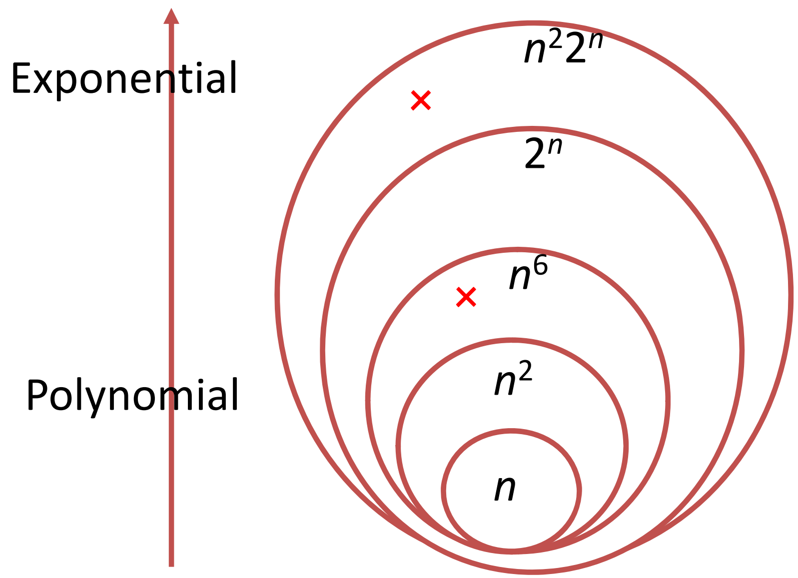
5. Computational Complexity

5.1. Measuring Computation Time

5.1.3. Time complexity of a problem

> Definition: For a function $t(n)$ over natural numbers, the set of all sets (i.e. recognition problems) with time complexities $O(t(n))$ is called ***$O(t(n))$*-time complexity class**, and it is denoted by TIME($t(n)$). Such a function $t(n)$ is called a time limit.

Ex. 1 PRIME was in TIME($n^2 2^n$),
but now it is in TIME(n^6).



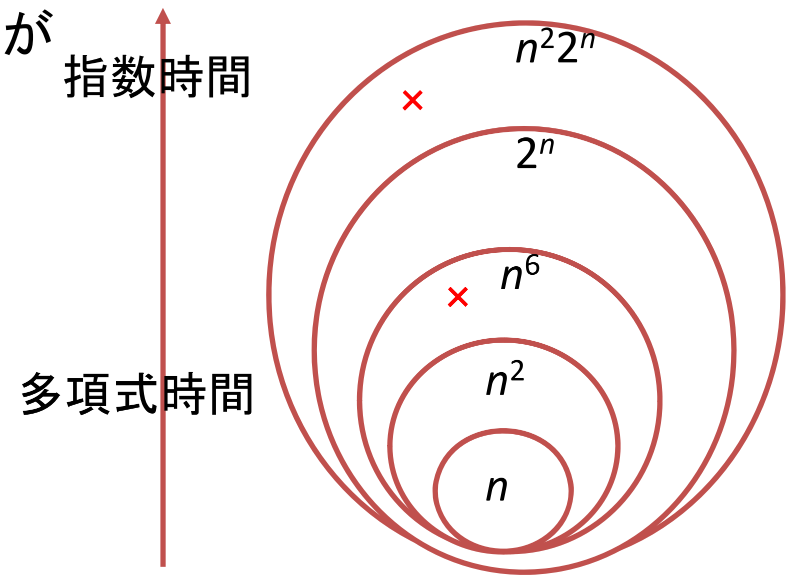
5.計算量の理論

5.1.計算時間の評価

5.1.3.問題の時間計算量

定義: 自然数上の関数 $t(n)$ に対して，時間計算量 $O(t(n))$ の集合(認識問題)全体の集合を ***O(t(n))*時間計算量クラス** とよび，TIME($t(n)$) とかく．こうした関数 $t(n)$ は制限時間と呼ぶ．

例1 PRIME は TIME($n^2 2^n$) の要素であったが今は TIME(n^6) の要素．



5. Computational Complexity

5.2. Time Complexity Classes

5.2.1. Representative time complexity classes

$$\mathcal{P} \equiv \bigcup_{p:\text{polynomial}} \mathrm{TIME}(p(n))$$

$$\mathcal{E} \equiv \bigcup_{c>1} \mathrm{TIME}(2^{cn})$$

$$\mathcal{EXP} \equiv \bigcup_{p:\text{polynomial}} \mathrm{TIME}(2^{p(n)})$$

[Note]
In these classes, polynomial time difference is not matter.
(We do not mind the difference of machine models.)

$\mathcal{C}$ set : set in the complexity class $\mathcal{C}$.
$\mathcal{C}$ problem : problem of recognizing a $\mathcal{C}$ set.

Problems not in $\mathcal{P}$ are intractable from the practical viewpoint…

5.計算量の理論

5.2. 時間計算量クラス

5.2.1. 代表的な時間計算量クラス

$$\mathcal{P} \equiv \bigcup_{p:\text{多項式}} \mathrm{TIME}(p(n))$$

$$\mathcal{E} \equiv \bigcup_{c>1} \mathrm{TIME}(2^{cn})$$

$$\mathcal{EXP} \equiv \bigcup_{p:\text{多項式}} \mathrm{TIME}(2^{p(n)})$$

[注意]
これらのクラスでは，多項式程度の違いは問題にならない．(マシンモデルによる違いは気にしないでよい．)

$\mathcal{C}$ 集合：計算量クラス $\mathcal{C}$ に属する集合．
$\mathcal{C}$ 問題：$\mathcal{C}$ 集合の認識問題．

$\mathcal{P}$ に入らない問題は現実的には手におえない．．．

5. Computational Complexity

5.2. Time Complexity Classes

5.2.1. Representative time complexity classes

Theorem: (1) $\mathcal{P} = \bigcup_{c>0} \mathrm{TIME}(n^c)$, (2) $\mathcal{EXP} = \bigcup_{c>0} \mathrm{TIME}(2^{n^c})$

Proof (Outline):

$\mathcal{T}_1$: set of polynomials of the form of n^c.

$\mathcal{T}_2$: set of all polynomials

→ since $\mathcal{T}_1 \subseteq \mathcal{T}_2$, $\mathrm{TIME}(\mathcal{T}_1) \subseteq \mathrm{TIME}(\mathcal{T}_2)$

p: arbitrary polynomial (p is any element of $\mathcal{T}_2$)

If the maximum degree of a polynomial p is k, $p(n) = O(n^k)$

Thus, $\mathrm{TIME}(p(n)) \subseteq \mathrm{TIME}(n^k) \subseteq \mathrm{TIME}(\mathcal{T}_1)$

Therefore, $\mathrm{TIME}(\mathcal{T}_1) = \mathrm{TIME}(\mathcal{T}_2)$

Q.E.D.

5.計算量の理論

5.2.時間計算量クラス

5.2.1.代表的な時間計算量クラス

定理: (1) $\mathcal{P} = \bigcup_{c>0} \mathrm{TIME}(n^c)$, (2) $\mathcal{EXP} = \bigcup_{c>0} \mathrm{TIME}(2^{n^c})$

証明 (概略):
T_1: n^c という形の多項式の集合.
T_2: すべての多項式の集合.
→ $T_1 \subseteq T_2$ より, $\mathrm{TIME}(T_1) \subseteq \mathrm{TIME}(T_2)$
p: 任意の多項式 (p はT_2の任意の要素)
多項式 p の最大次数をkとすると, $p(n) = O(n^k)$
よって $\mathrm{TIME}(p(n)) \subseteq \mathrm{TIME}(n^k) \subseteq \mathrm{TIME}(T_1)$
したがって $\mathrm{TIME}(T_1) = \mathrm{TIME}(T_2)$

Q.E.D.

5. Computational Complexity

5.2. Representative Time Complexity Classes

5.2.2. Representative problems and their complexity

5.2.2.1. Problem of evaluating propositional expression(PROP-EVAL)

Input : $<F, < a_1, a_2, \dots , a_n >>$

- F is an extended propositional expression
- $(a_1, a_2, \dots , a_n)$ is a truth assignment to F

Question : $F(a_1, a_2, \dots , a_n) = 1$?

(x,y)	$x \rightarrow y$ $(\neg x \vee y)$	$x \leftrightarrow y$ $((x \rightarrow y) \wedge (y \rightarrow x))$
(0,0)	1	1
(0,1)	1	0
(1,0)	0	0
(1,1)	1	1

5.計算量の理論

5.2. 代表的な時間計算量クラス

5.2.2. 代表的な問題とその計算量

5.2.2.1. 命題論理式の評価 (PROP-EVAL)

入力：$\langle F, \langle a_1, a_2, \ldots, a_n \rangle\rangle$

- F は拡張命題論理式
- $(a_1, a_2, \ldots, a_n)$ は F への真偽値割当て

質問： $F(a_1, a_2, \ldots, a_n) = 1$?

(x,y)	$x \rightarrow y$ $(\neg x \vee y)$	$x \leftrightarrow y$ $((x \rightarrow y) \wedge (y \rightarrow x))$
(0,0)	1	1
(0,1)	1	0
(1,0)	0	0
(1,1)	1	1

5. Computational Complexity

5.2. Representative Time Complexity Classes

5.2.2. Representative problems and their complexity

5.2.2.1. Problem of evaluating propositional expression (PROP-EVAL)

Input : $<F, < a_1, a_2, \dots , a_n >>$

- F is an extended prop. expression
- $(a_1, a_2, \dots , a_n)$ is a truth assignment to F

Question : $F(a_1, a_2, \dots , a_n) = 1$?

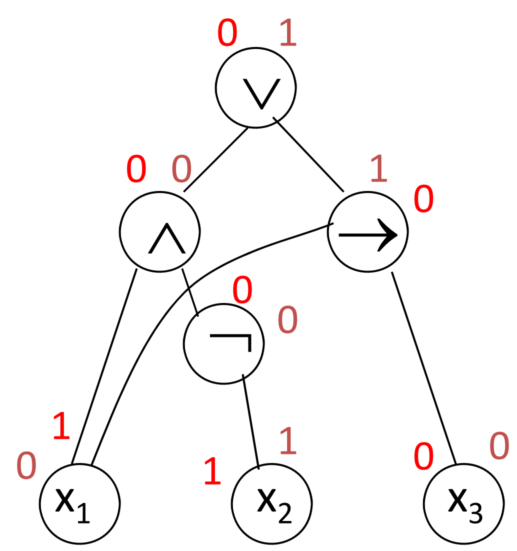
PROP-EVAL $\in \mathcal{P}$

Construct a *computation tree* from a code of F.
It is built in time $O(|F|^3)$.
Once computation tree is built,
we can easily obtain the value
$F(a_1, a_2, \dots , a_n)$ in a bottom-up fashion.

$F(0,1,0)=1$

$F(1,1,0)=0$

Ex. : $F(x_1, x_2, x_3) = [x_1 \wedge \neg x_2] \vee [x_1 \rightarrow x_3]$



computation tree

5.計算量の理論

5.2.代表的な時間計算量クラス

5.2.2.代表的な問題とその計算量

5.2.2.1.命題論理式の評価(PROP-EVAL)

入力：$<F, < a_1, a_2, \ldots , a_n >>$

- F は拡張命題論理式
- $(a_1, a_2, \ldots , a_n)$は F への真偽値割当て

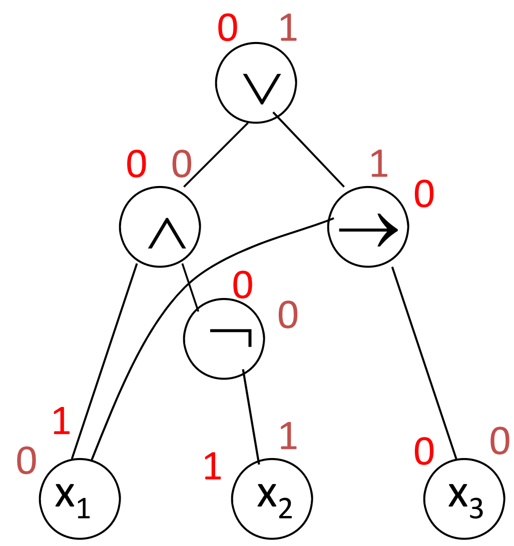
質問： $F(a_1, a_2, \ldots , a_n) =1$?

$\boxed{\text{PROP-EVAL} \in \mathcal{P}}$

F のコードから計算木を作る.
構築に必要な時間は $O(|F|^3)$.
ひとたび計算木ができると,
ボトムアップに計算すると
$F(a_1, a_2, \ldots , a_n)$ の値は簡単に求まる.

$F(0,1,0)=1$

$F(1,1,0)=0$



計算木

Ex.： $F(x_1, x_2 , x_3) = [x_1 \wedge \neg\, x_2] \vee [x_1 \rightarrow x_3]$

5. Computational Complexity

5.2. Representative Time Complexity Classes

5.2.2. Representative problems and their complexity

5.2.2.2. Satisfiability (SAT)

Input : $<F>$ F is 2-conjunctive normal form

Question : Is there any assignment such that $F(a_1, a_2, \ldots, a_n) = 1$?

Conjunctive Normal Form (CNF)

$$F = (\bigcirc \vee \bigcirc \vee \ldots \vee \bigcirc) \wedge (\bigcirc \vee \ldots \vee \bigcirc) \wedge \ldots \wedge (\ldots)$$

- described by $\wedge$ of $\vee$ of literals.

k SAT: Each closure contains *k* literals

exactly/at most

We can define 3SAT, 4SAT similarly.

SAT consists of any CNF.

ExSAT consists of any extended propositional expression.

5.計算量の理論

5.2.代表的な時間計算量クラス

5.2.2.代表的な問題とその計算量

5.2.2.2. 充足可能性 (SAT)

入力 : <*F*> *F* は 2-和積標準形
質問 : $F(a_1, a_2, \ldots, a_n) = 1$とする真偽値割当てが存在するか？

和積標準形 (CNF)

$F = (\bigcirc \vee \bigcirc \vee \ldots \vee \bigcirc) \wedge (\bigcirc \vee \ldots \vee \bigcirc) \wedge \ldots \wedge (\ldots)$

- リテラルの∨の∧で表現される式

k SAT: 各項が *k* 個のリテラルを含む

ちょうど/たかだか

3SAT, 4SATも同様に定義できる.

SAT は任意の CNF を許す

ExSAT は拡張命題論理式（∨,∧,→,↔）を許す

5. Computational Complexity

5.2. Representative Time Complexity Classes

5.2.2. Representative problems and their complexity

5.2.2.3. Graph reachability problem (ST-CON)

Input : $\langle G,s,t \rangle$: an undirectd graph G, $1 \leqq s,t \leqq n(=|G|)$

Question : Does G have a path from s to t?

5.2.2.4. Euler cycle problem (DEULER)

Input : $\langle G \rangle$: a directed graph G

Question : Does G have an Euler cycle?

5.2.2.5 Hamiltonian cycle problem (DHAM)

Input : $\langle G \rangle$: a directed graph G

Question : Does G have a Hamiltonian cycle?

Actually,
directed/undirected
cycle/path
do not matter

- Cycle is a path that shares two endpoints.
- Euler cycle is a cycle that visits all edges once.
- Hamiltonian cycle is a cycle that visits all vertices once.

5.計算量の理論

5.2.代表的な時間計算量クラス

5.2.2.代表的な問題とその計算量

5.2.2.3. グラフの到達可能性問題 (ST-CON)

入力 : <*G*,*s*,*t*> : 無向グラフ *G*, $1 \leqq s,t \leqq n(=|G|)$

質問 : *G* は *s* から *t* への経路を持つか？

5.2.2.4. オイラー閉路問題 (DEULER)

入力 : <*G*>: 有向グラフ *G*

質問 : *G* はオイラー閉路を持つか？

実際には,
有向/無向
閉路/パス
という違いは問題にならない

5.2.2.5 ハミルトン閉路問題 (DHAM)

入力 : <*G*>: 有向グラフ*G*

質問 : *G* はハミルトン閉路を持つか？

- 閉路とは両端点を共有する経路.
- オイラー閉路とはすべての辺をちょうど一回通る閉路.
- ハミルトン閉路とはすべての頂点をちょうど一回通る閉路.

5. Computational Complexity

5.2. Representative Time Complexity Classes

5.2.2. Representative problems and their complexity

It is known that :

- The following problems are in $\mathcal{P}$:
  - ✓PROP-EVAL, 2SAT, ST-CON, DEULER
- The following problems are in $\mathcal{E}$, but…
  - ✓3SAT, DHAM

The class $\mathcal{NP}$ between $\mathcal{P}$ and $\mathcal{E}$?

5.計算量の理論

5.2.代表的な時間計算量クラス

5.2.2.代表的な問題とその計算量

以下は既知:

- 以下の問題は $\mathcal{P}$ の要素:
  - ✓PROP-EVAL, 2SAT, ST-CON, DEULER
- 以下の問題は $\mathcal{E}$ の要素だが…
  - ✓3SAT, DHAM

$\mathcal{P}$ と $\mathcal{E}$ の間(?)にあるクラス $\mathcal{NP}$