Note Title 6/18/2010

Computational Complexity

or

Theory of Complexity

By any objective standard, the theory of computational complexity ranks as one of the greatest intellectual achievements of humankind -- along with fire, the wheel, and computability theory. That it isn't taught in high schools is really just an accident of history.

To most people who are not theoretical computer scientists, the theory of computational complexity–one of the great intellectual achievements of the twentieth century–is simply a meaningless jumble of capital letters.

Scott Aaronson [2]

Theoretical Computer Science ⟹ Computational Complexity

هدف تئوری نظریه: دسته‌بندی مسایل بر اساس میزان منابعی که برای حل آنها لازم است.

مسئله A از مسئله B سخت‌تر است اگر A بیشتر از B منابع مصرف کند.

منابع اصلی:
- TIME
- SPACE (یا حافظه)

problem → یک خانواده بی‌نهایت عضوی

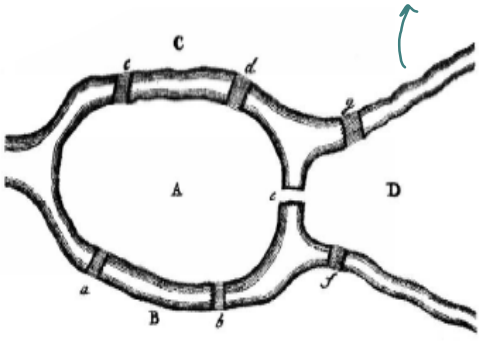
$$\text{Problem} = \{ P_1, P_2, \ldots \}$$

P_1: یک نمونه از مسئله (Instance)

مثال:

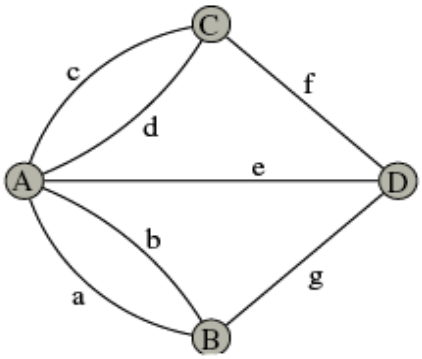
- کونیگزبرگ — Königsberg
- کالینینگراد — Kaliningrad

رودخانه Pregel



آیا مسیری وجود دارد که از تمام پل‌ها یک و فقط یک بار عبور کند؟

• مسئله (1736)



Eulerian Path problem.

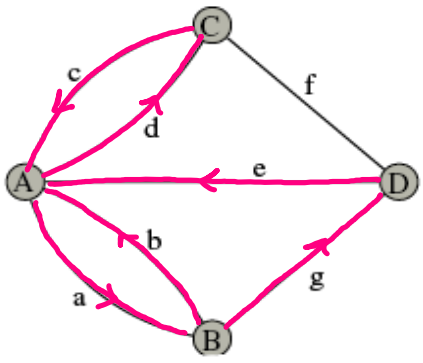
داده: $G = (V, E)$ graph

مسئله: آیا مسیری وجود دارد که از تمام اضلاع یک و فقط یک بار عبور کند؟

• **تعریف:** هر گاه مسیر شما هم باشد، یک حلقهٔ اویلری Eulerian loop خوانده می‌شود.

• **تعریف:** گرافی که مسیر یا حلقهٔ اویلری داشته باشد، گراف اویلری خوانده می‌شود.

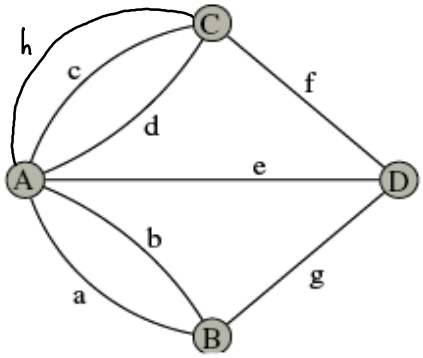
• **قضیه (اویلر):** یک گراف، گراف اویلری است اگر و فقط اگر تعداد رأس‌های با درجهٔ فرد آن یا ۰ باشد یا ۲. (۰ ← حلقهٔ اویلری، ۲ ← مسیر اویلری)



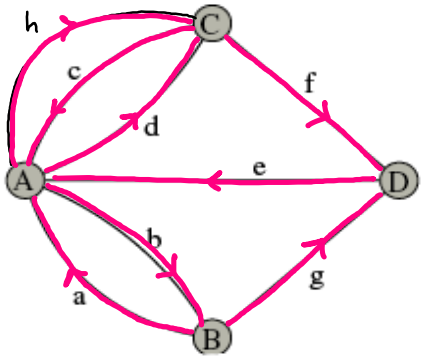
شروع

این گراف اویلری نیست. ←

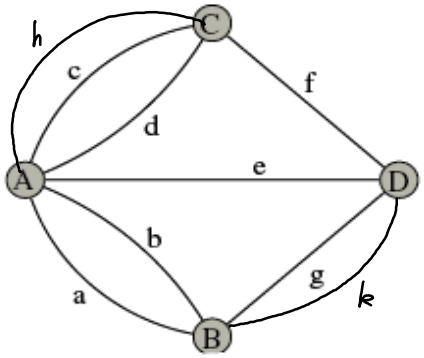
این گراف اویلری است ⟵



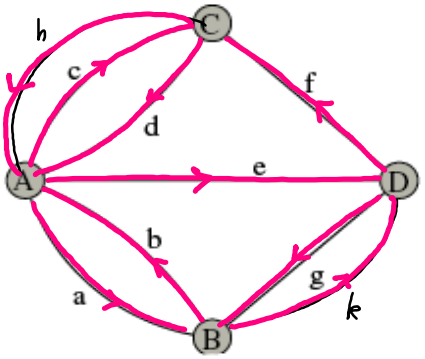
g e b a d c h f



این گراف یک حلقه اویلری دارد. ⟵



k g b e f d c h a



- **الگوریتم:** مجموعه‌ای از مراحل مشخص که بتواند حل مسئله را برای هر نمونه‌ای از مسئله تولید کند (از جمله برای بدترین نمونه‌ی آن). (WORST CASE)

- **میزان مصرف منابع:**

  - TIME
  - SPACE (or Memory)

- برای بررسی یک مسئله، ما به دنبال بهترین و سریع‌ترین الگوریتم آن که تا حالا بلد نیستیم.

↓

برای بررسی یک مسئله، حد بالای مصرف منابع را بدانیم.

نظریه پیچیدگی اساساً یک نظریه برای حدی بالاست.

Time Complexity. ← از این به بعد در مبحث زمان تنها می گوییم.

$P = \{ x_1, x_2, x_3 \dots \}$ = problem.

$x(n)$ = نمونه ای از مسئله که طول آن n است

$$T(n) = \max_{|x| = n} T(x)$$

- کدام زمان مورد نظر است؟

T_1 ؟ زمان سامت دیواری برحسب ثانیه

T_2 ؟ زمان یک کرنومتر برحسب دهم ثانیه

T_3 ؟ زمان cpu برحسب 10^{-9} ثانیه

$$T_1(n) = 10\, T_2(n) = 10\, T(n)$$

$T(n) = O(f(n))$ $\quad \exists\, c, n_0 \mid \forall\, n \geq n_0 \quad T(n) \leq c\, f(n)$

$T(n) = \Theta(f(n))$ $\quad \exists\, c_1, c_2, n_0 \mid \forall\, n \geq n_0 \quad c_1 f(n) \leq T(n) \leq c_2 f(n)$

مثال: ضرب Multiptication

$x = a_{n-1} a_{n-2} \dots a_0$

$y = b_{n-1} b_{n-2} \dots b_0$ $\quad T(n) = \Theta(n^2)$.

ایده الگوریتم تقسیم:

$x = 2^{n/2} a + b$

$y = 2^{n/2} c + d$ $\quad a, b, c, d = \frac{n}{2}$ digit numbers.

$$xy = 2^n ac + 2^{n/2}(ad + bc) + bd$$

ضرب ۲ عدد n رقمی ≡ ضرب ۴ عدد $\frac{n}{2}$ رقمی + جمع اعداد $\frac{n}{2}$ رقمی.

$$\rightarrow T(n) = 4T(\tfrac{n}{2}) + \Theta(n)$$

$$\rightarrow T(n) = 2^2 T(\tfrac{n}{2}) + \Theta(n)$$

$$\vdots$$

$$T(n) = 2^4 T(\tfrac{n}{2^2}) + \Theta(n)$$

$2^k = n \rightarrow 2^{2k} = n^2$

$$T(n) = \Theta(n^2)$$

• یک الگوریتم بهتر

به ad , bc بصورت جداگانه نیاز نداریم: ←

$$(a+b)(c+d) = ac + bd + ad + bc$$

کافی است که ۳ تا عمل ضرب بزنیم تا $ad+bc$ پیدا شود. ←

$$T(n) = 3\,T\left(\frac{n}{2}\right) + \theta(n)$$

$$\vdots$$

$$T(n) = 3^2\,T\left(\frac{n}{2^2}\right) + \theta(n)$$

$2^k = n \qquad 3^k = n^x \quad \Rightarrow \quad k \ln 2 = \ln n$

$k \ln 3 = x \ln n$

$$\rightarrow \quad x = \frac{\ln 3}{\ln 2} = 1.58$$

$$T(n) = \theta(n^{1.58}).$$

کلاس P : مسائل قابل حل.

اگر $f(n)$ یک تابع مشخص باشد در زمان است.

$\mathrm{TIME}(f(n)) =$ کلاس مسائلی که در زمان $O(f(n))$ حل می‌شوند.

$\mathrm{TIME}(n^k) =$ کلاس مسائلی که در زمان $O(n^k)$ حل می‌شوند.

- مثال:

$\mathrm{Multiplication} \in \mathrm{TIME}(n^2)$

$\mathrm{Multiplication} \notin \mathrm{TIME}(n)$

- $\ldots \mathrm{TIME}(n^k) \subset \mathrm{TIME}(n^{k+1}) \ldots$

$\mathrm{Euler} \in \mathrm{TIME}(n^2)$

- کلاس P

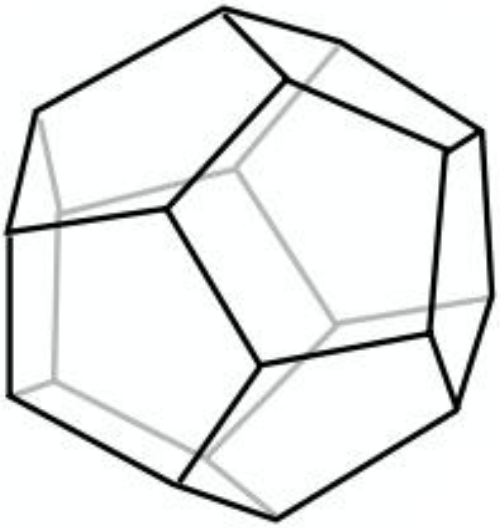
$$P := \bigcup_{k \geq 0} \mathrm{TIME}(n^k)$$

مسائل P مسائلی هستند که در مورد حل آنها یک کامپیوتر

استدلال وجود دارد.

کلاس NP : مسایل دشوار

۱۲ دی ۱۴۰۲ ، رأس ۲۰ ، ضلع ۳۰



مسئله مسیر همیلتونی (1859)

آیا مسیری وجود دارد که از هر رأس تنها یک بار عبور کند؟

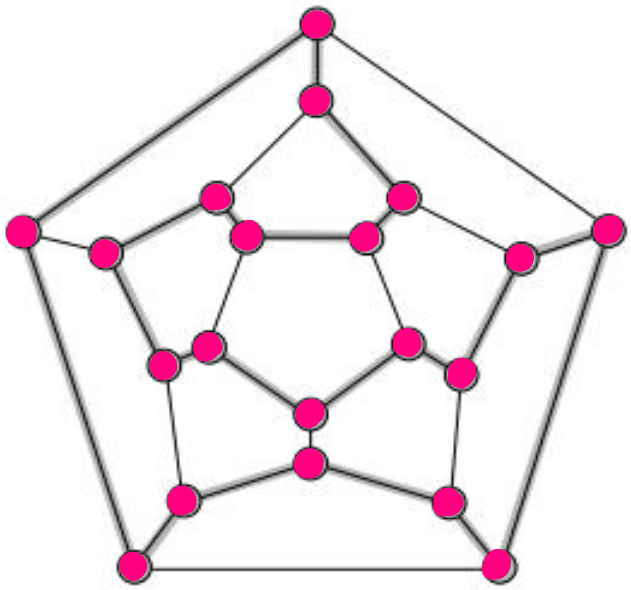
مسایل کلاس NP ، مسایلی هستند که یافتن حل آنها دشوار و استدلالی مشکل و تنها با جستجوی کامل همه امکانات ممکن هستیم.

تشابه با مسئله

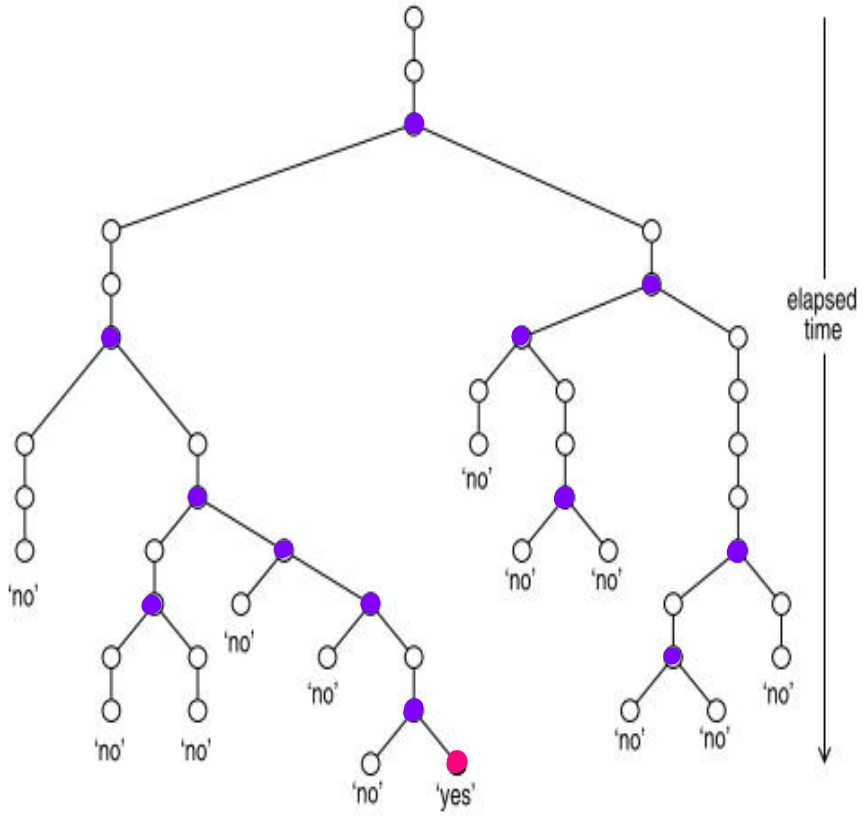
پیدا کردن سوزن در کاه‌دان.

پیدا کردن = دشوار

تایید کردن = آسان



- یک مسئله تصمیم‌گیری در کلاس NP است، اگر و فقط اگر در زمان polynomial توسط یک الگوریتم غیرقطعی حل شود.



با توجه به تعریف و دیاگرام بالا ←

$P \subseteq NP$

در رابطه بالا، مسایل P را منحصر کردیم به مسایل تصمیم‌گیری.

• یک تعریف معادل از مسئله NP.

یک مسئله تصمیم گیری X متعلق به NP است اگر در مورد هر مسئله نمونه از آن، یک اثبات قابل تست در زمان چند جمله‌ای وجود دارد.

• مثال: مسئله گراف همیلتونی از این نوع است. زیرا اگر کسی تعریف یک گراف G همیلتونی است به معنی این است که یک سیر همیلتونی در G نشان دهد. تست کردن این سیر زمان چند جمله‌ای خواهد.

• مثال: مسئله مرکب بودن Compositeness یک مسئله NP است.

• یک سوال مهم: آیا مسئله‌ای وجود دارد که متعلق به NP نباشد؟ بله!

• مثال: آیا این گراف G ناهمیلتونی (Non-Hamiltonian) است؟

واضح است که پاسخ مثبت به این سوال را نمی‌توان در زمان چند جمله‌ای تست کرد.

• مثال: آیا عدد N اول است؟ primality

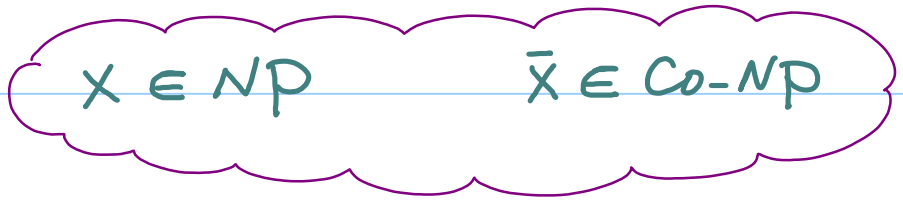
• تعریف: کلاس Co-NP یک مسئله تصمیم‌پذیر که بتوان پاسخ منفی آن را با یک اثبات تأیید کرد در زمان چندجمله‌ای و مجموعه‌ی همه‌ی مسائل متعلق به کلاس Co-NP یک Co-NP است.

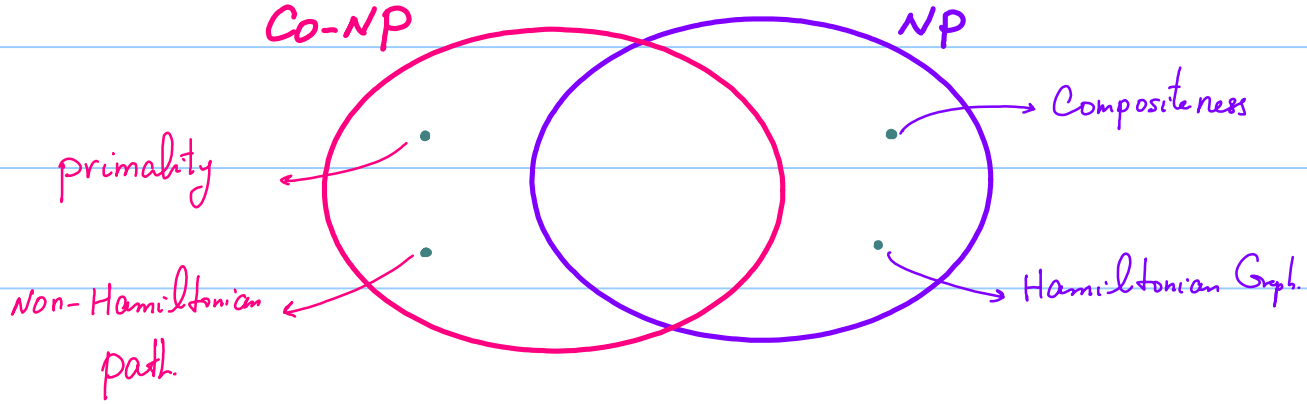
Hamiltonian Graph $\in$ NP

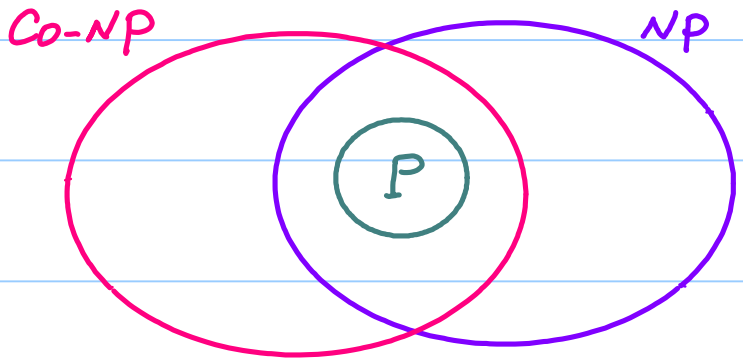
Compositeness $\in$ NP

Non-Hamiltonian Graph $\in$ Co-NP

primality $\in$ Co-NP







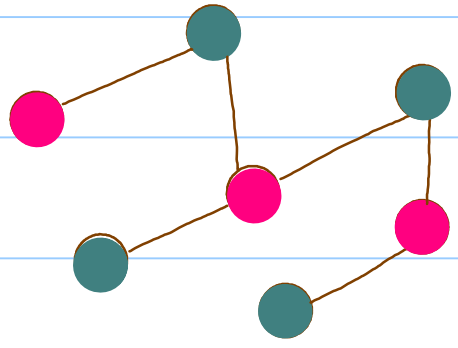
$$P \subset NP \cap Co\text{-}NP$$

کلاس NP شامل تعدادی از مسایل بسیار مهم و پیچیده است.

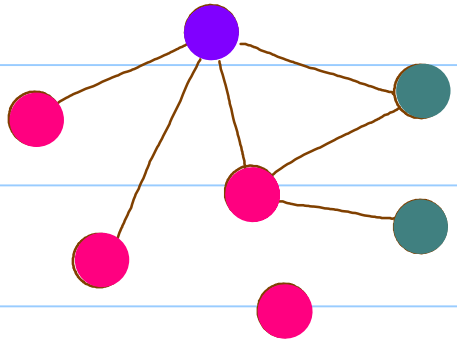
- مثال ۱) k-Coloring problem

Input : a graph $G=(V,E)$, k

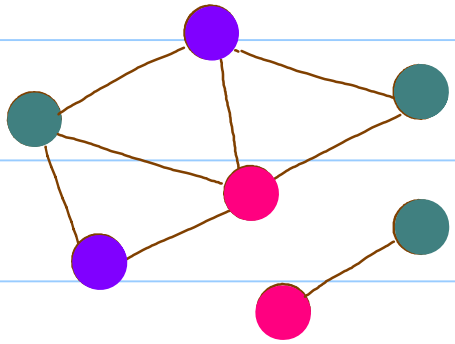
question : آیا می‌توان با **حداکثر** k رنگ گراف را رنگ‌آمیزی کرد طوری که هیچ دو رأس مجاور رنگ یکسان نداشته باشند.



2-Colorable



3- Colorable

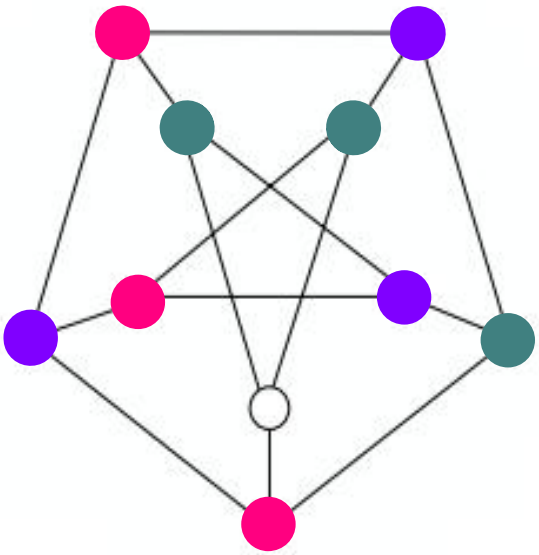


3- Colorable

بسیاری از مسایل را می‌توان در قالب k-رنگ‌پذیری مطالعه کرد.

مثال: در یک کنفرانس که شرکت کنندگان ممکن است به بیشتر از یک کنفرانس علاقه‌مند باشند، می‌خواهیم با حداقل زمان ممکن کنفرانس‌ها را ترتیب دهیم طوری که هیچ کس در کنفرانس مورد علاقه‌اش شرکت نکند.

فرض این است که سالن کنفرانس به اندازه کافی در اختیار داریم.



هر کنفرانس = یک راس

اتصال بین دو کنفرانس = وجود فرد یا افرادی که مایل به شرکت در هر دو کنفرانس هستند.

تعداد رنگ‌ها = # Time slots

1- Coloring $\in P$

2- Coloring $\in P$

$k \geq 3$ Coloring $\in NP$

The 2-SAT problem.

$\phi(x_1, x_2, \dots x_N) = \bigwedge_i C_i$ $\qquad C = (x_j^{\pm} \vee x_k^{\pm})$

Input: $\phi(x_1, x_2, \dots x_N)$

question: $\phi(x_1, \dots x_N) = 1$ برای $(x_1, x_2, \dots x_N)$ آیا

(i.e: Is ϕ satisfiable?)

The k-SAT problem

Each Clause $C = (x_{i_1}^{\pm} \vee x_{i_2}^{\pm} \vee \dots x_{i_k}^{\pm})$

$1\text{-SAT}, 2\text{-SAT} \in P.$

$k\text{-SAT} \in NP \quad k \geq 3.$

Reduction کاهش

- مثال:

$SAT < 3\text{-SAT}$

- اگر مسئله 3-SAT حل شود، می‌توان مسئله SAT را نیز حل کرد.
- مسئله SAT در زمان چند جمله‌ای مسئله 3-SAT کاهش‌پذیر است.
- مسئله SAT از مسئله 3-SAT کمتر است.
- الگوریتم SAT می‌تواند به عنوان یک زیربرنامه (Subroutine) از الگوریتم 3-SAT استفاده کند.

اثبات: یک عبارت C در SAT در نظر می‌گیریم:

if $C = l_1 \longrightarrow C' := l_1 \vee l_1 \vee l_1$

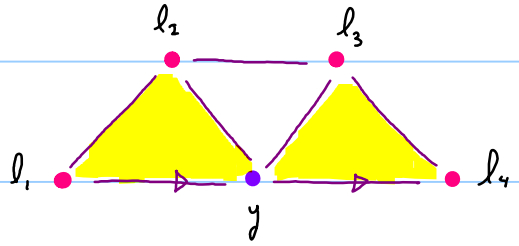
if $C = l_1 \vee l_2 \longrightarrow C' := l_1 \vee l_2 \vee l_1$

if $C = l_1 \vee l_2 \vee l_3 \vee l_4 \longrightarrow C' := (l_1 \vee l_2 \vee y) \wedge (\bar{y} \vee l_3 \vee l_4)$

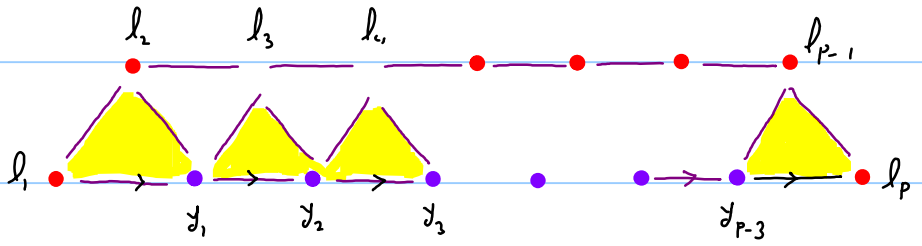
اگر C' ارضا پذیر $\Leftarrow$ $l_1 \vee l_2 \vee y$ و $\bar{y} \vee l_3 \vee l_4$ هر دو ارضا هستند.

اگر $y=0$ $\leftarrow$ ارضا پذیر $l_1 \vee l_2$ $\leftarrow$ ارضا پذیر C $\leftarrow$ ارضا پذیر.

اگر $y=1$ $\leftarrow$ ارضا پذیر $l_3 \vee l_4$ $\leftarrow$ ارضا پذیر C $\leftarrow$ ارضا پذیر.



if $C = l_1 \vee l_2 \vee l_3 \vee l_4 \vee l_5 \vee \dots\dots\dots \vee l_p$



$$C' = (l_1 \vee l_2 \vee y_1) \wedge (\bar{y}_1 \vee l_3 \vee y_2) \wedge (\bar{y}_2 \vee l_4 \vee y_3) \wedge \dots\dots (y_{p-4} \vee l_{p-2} \vee \bar{y}_{p-3}) \wedge (y_{p-3} \vee l_{p-1} \vee l_p)$$

تعریف : $P < Q$ اگر حل Q در زمان چندجمله‌ای حل P بگیرد.

تاکنون مثال‌های زیادی از کاهش مسئله ارائه و اثبات شده‌اند.

$k\text{-SAT} < 3\text{-SAT}$

$k\text{-Coloring} < 3\text{-Coloring}$

$3\text{-SAT} < 3\text{-Coloring}$

$3\text{-Coloring} < \text{Hamiltonian path}$

اولین قضیه در مورد کاهش توسط Stephen Cook در ۱۹۷۱ اثبات شد.

$\forall P \in NP \quad , \quad P < 3\text{-SAT}$

یک الگوریتم چندجمله‌ای برای 3-SAT

$P = NP$

تعریف: یک مسئله p، خانواده‌ی NP-Complete است از

الف) $p \in NP$

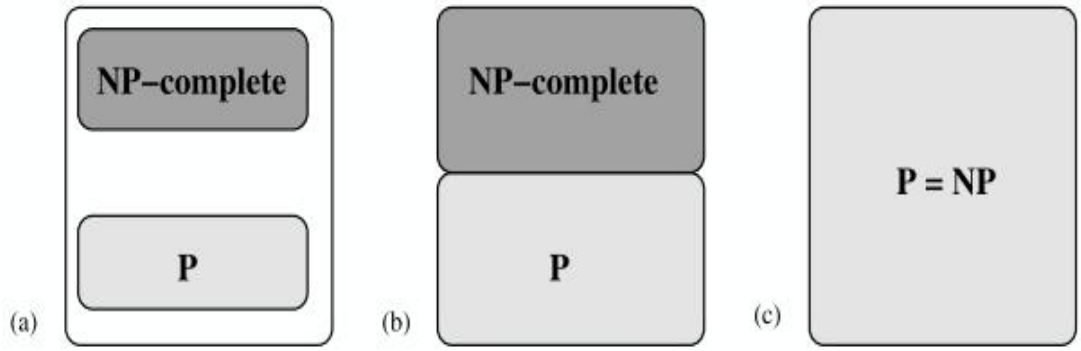
ب) $\forall Q \in NP, \quad Q \leq p.$

یعنی برای یک مسئله‌ی جدید لازم است که:

NP-Complete خواهد بود.

$NP\text{-Complete} = \{ 3\text{-SAT}, \text{Hamiltonian-path}, 3\text{-coloring}, \dots \}$

کدام تصویر درست است؟

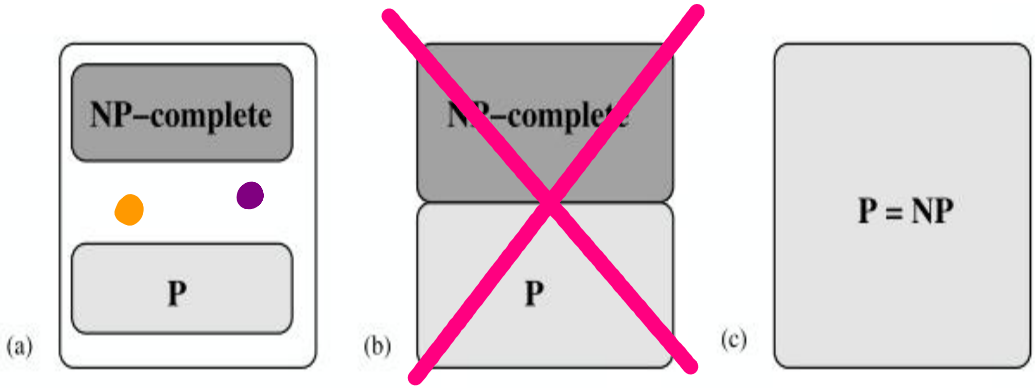


قضیه یک میلیون دلاری Clay Institute

قضیه: اگر $P \neq NP$، آنگاه

$\exists Q \mid Q \notin NP\text{-complete}, Q \notin P$

نیاز بریک



- Factoring
- Graph Isomorphism

Factoring

- Input : دو عدد صحیح N و M
- question : آیا N یک فاکتور کوچک‌تر از M دارد؟

Graph Isomorphism

- Input : $G=(V,E)$ و $G'=(V',E')$
- question : آیا تناظری بین V و V' وجود دارد که دو گراف یکی کند؟

چرا سوال P=NP مهم است؟

proof check

- **Input:** A یک مجموعه از اصول موضوع، S یک گزاره یا قضیه، P یک اثبات
- **Question:** آیا P یک اثبات معتبر برای گزاره S هست؟

این الگوریتم در زمان چندجمله‌ای کار می‌کند. بنابراین proof check $\in P$.

این نتیجه بابت این است که یک الگوریتم دیگر متعلق به NP باشد.

proof-existence:

- **Input:** A یک مجموعه از اصول موضوع، S یک گزاره یا قضیه، l یک عدد صحیح
- **question:** آیا برای S یک اثبات P با طول کمتر از l وجود دارد.

proof check $\in P \Rightarrow$ Proof-existence $\in NP$

if $NP = P \rightarrow$ Proof check $\in P$

↓

تکلیف هر قضیه‌ای، هرگزاره‌ای، هر حدسی مثل حدس گلدباخ یا حدس ریمان،

نظیرآن خیلی زود روشن می‌شود.

↓

Mathamatics ≡ Computer programming

Thinking and Creativity ≡ fast Computation ?

- Decision Problems
- optimization problems.

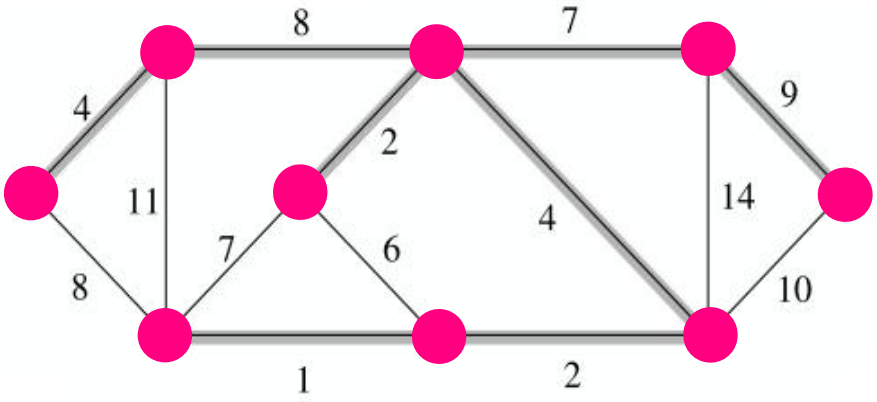
- مسایل تصمیم‌پذیر: آیا در این گراف راه‌ای یک مسیر هست؟

آیا در این گراف راه‌ای یک مسیر هست؟

- مسایل بهینه‌سازی: کوتاه‌ترین تعداد گام در این گراف کدام است؟

آزمودن پاسخ ممکن است در زمان چند جمله‌ای ممکن باشد.

• مثال: مسئله درخت کمینه پوشان Minimum Spanning Tree



مثال:
یافتن یک شبکه
تلفن با کمترین
هزینه.

MST

Input: A weighted Graph $G=(V,E)$.

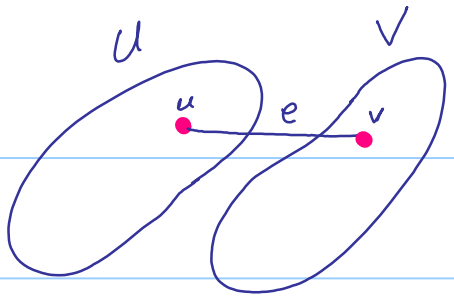
question: A Spanning Tree with minimum weight.

A Mathematical Insight ⟶ Polynomial Solution

درک ریاضی برای مسئله MST.

• قضیه: بازای هر زیرمجموعه U از V، ضلعی با کمترین وزن که یک سر آن در U و سر دیگر در $V-U$ است

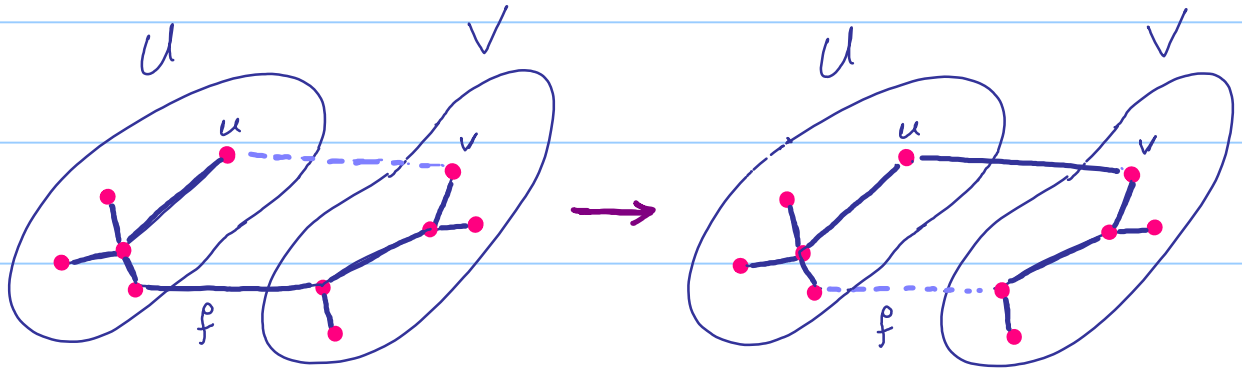
مربوط به جزء MST باشد.



- تقریباً احتیاج به اثبات ندارد

اثبات: فرض کنید e دارای کمترین وزن باشد

و جزء T نباشد. مثل این ←



$T \quad > \quad T'$

- نتیجه: با یک آگریدی حیص بیص می‌توان MST را حل کرد.

- مسئله فروشنده دوره‌گرد.

TSP

Input: An $N \times N$ matrix with $d_{ij} \geq 0$

question: A cyclic permutation $\pi := \begin{pmatrix} 1 & 2 & 3 & & N \\ \pi(1) & \pi(2) & & & \pi(N) \end{pmatrix}$ which

minimizes $\sum_{i=1}^{N} d_{i,\pi(i)}$

• مسائل بهینه‌سازی : optimization

$F = \{f_1, f_2, \dots\}$ = space of solutions

$c : F \longrightarrow R_+$ Cost function

- Optimization Problem : P(O) find f^* which minimizes c.
- Evaluation Problem : P(E) evaluate $c(f^*)$

بدست آوردن کمترین مقدار تابع f^* هدف.

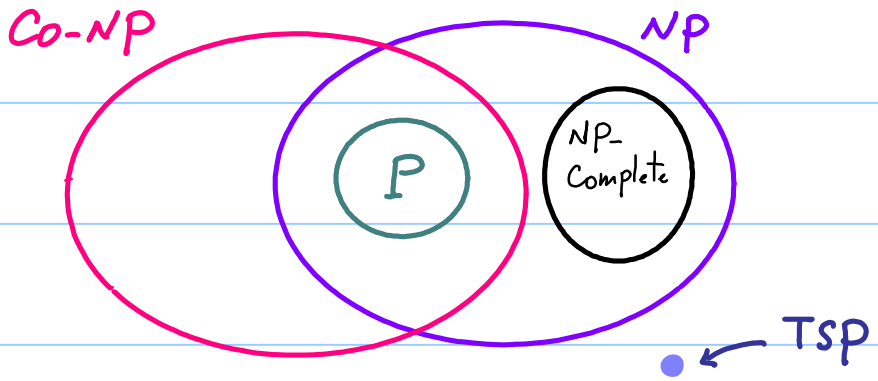
- Decision Problem : P(D) آیا f ای هست که $c < B$ باشد؟

$P(D) < P(E) < P(O)$.

• قضیه : TSP $\simeq$ Hamiltonian Path

Polynomial solution for TSP $\Longrightarrow$ P = NP.

Note : TSP $\notin$ NP.



مسایلی مثل TSP که NP هستند ولی حل آنها به معنی P = NP است

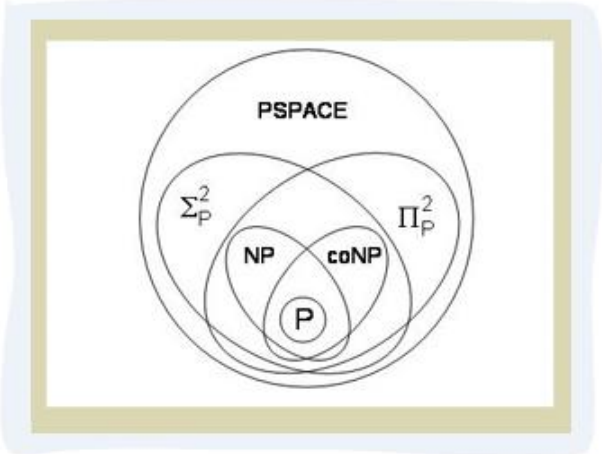
NP-HARD نامیده می‌شوند.

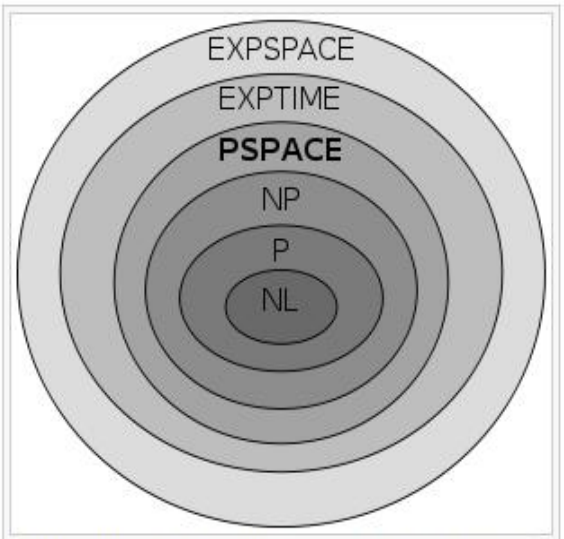
مسایل از نوع دیگر: مسایل شمارش

Counting Problems

چند تا پرکاه با طول N بین کاهدان وجود دارد؟

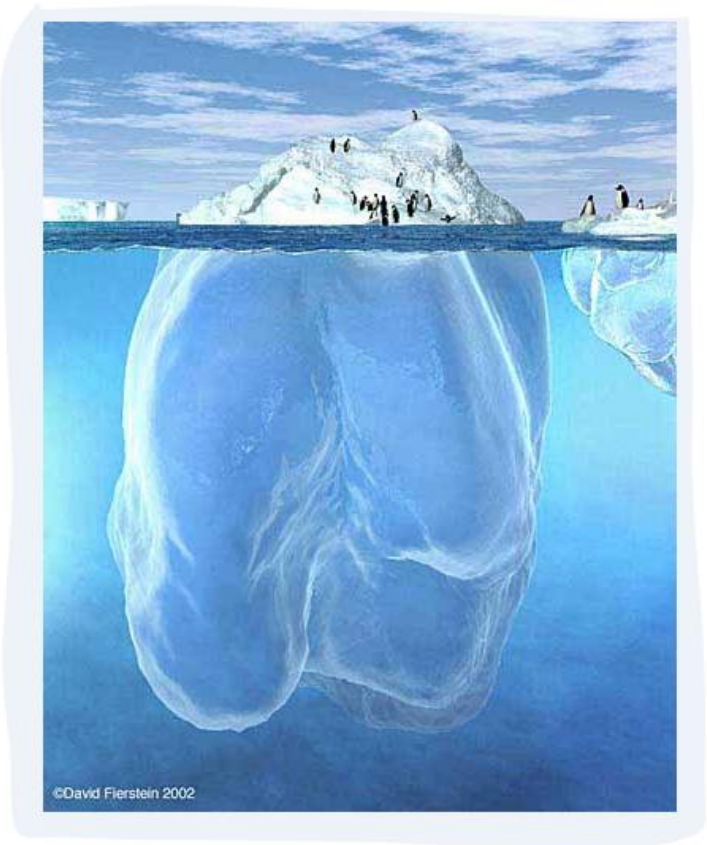
Complexity Zoo





A representation of the relation among complexity classes

Complexity class	Model of computation	Resource constraint
DTIME($f(n)$)	Deterministic Turing machine	Time $f(n)$
P	Deterministic Turing machine	Time poly(n)
EXPTIME	Deterministic Turing machine	Time $2^{poly(n)}$
NTIME($f(n)$)	Non-deterministic Turing machine	Time $f(n)$
NP	Non-deterministic Turing machine	Time poly(n)
NEXPTIME	Non-deterministic Turing machine	Time $2^{poly(n)}$
DSPACE($f(n)$)	Deterministic Turing machine	Space $f(n)$
L	Deterministic Turing machine	Space O(log n)
PSPACE	Deterministic Turing machine	Space poly(n)
EXPSPACE	Deterministic Turing machine	Space $2^{poly(n)}$
NSPACE($f(n)$)	Non-deterministic Turing machine	Space $f(n)$
NL	Non-deterministic Turing machine	Space O(log n)
NPSPACE	Non-deterministic Turing machine	Space poly(n)
NEXPSPACE	Non-deterministic Turing machine	Space $2^{poly(n)}$



- کلاس‌های دیگر:
- الگوریتم‌های تصادفی: