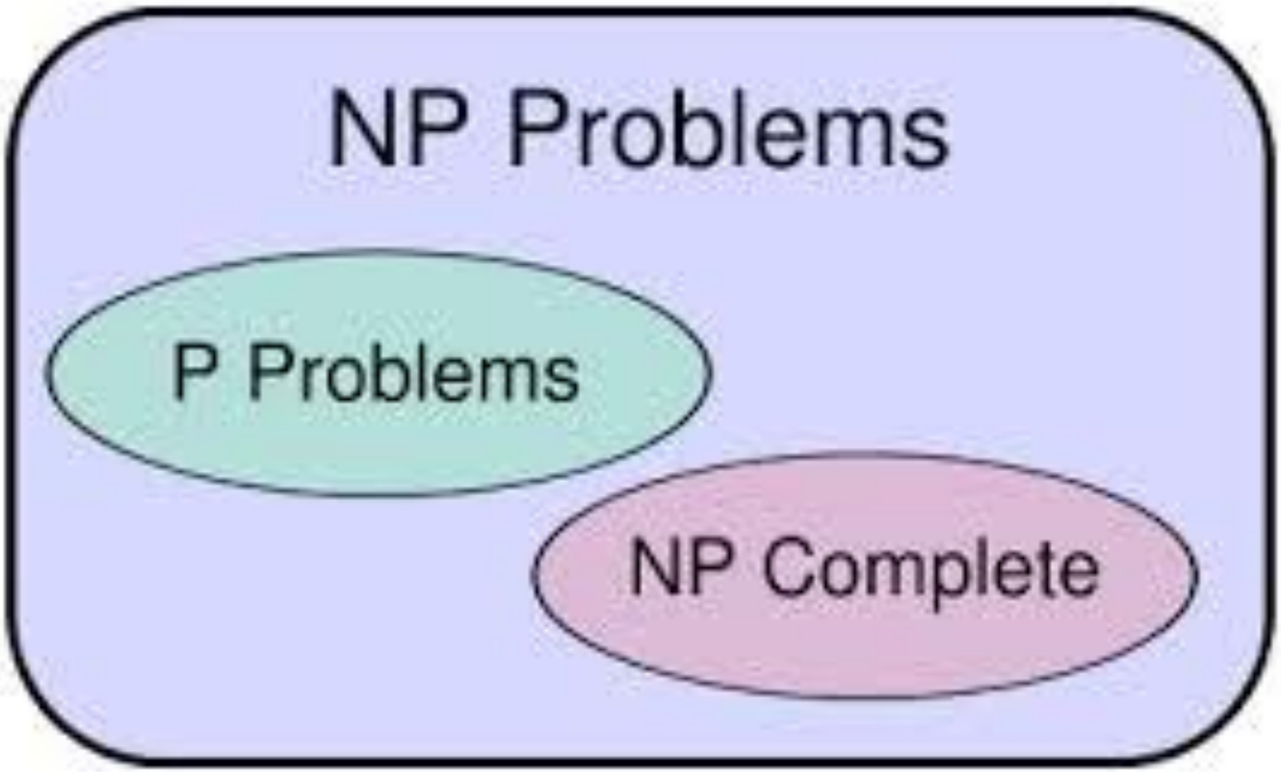
Teori P, NP, dan NP-Complete

Bahan Kuliah IF2211 Strategi Algoritma

Oleh: Rinaldi Munir
Program Studi Teknik Informatika ITB


NP Problems
P Problems
NP Complete

Pendahuluan

- Berdasarkan kebutuhan waktunya, algoritma untuk menyelesaikan persoalan dapat dibagi menjadi dua kelompok besar:

1. Algoritma waktu-polinom (*polynomial-time algorithms*)

2. Algoritma waktu-non-polinom (*nonpolynomial-time algorithms*)

- *Polynomial-time algorithm* adalah algoritma yang kempleksitas waktunya dibatasi oleh fungsi polinom terhadap ukuran masukannya (n).

  - Contoh:Persoalan *sorting* → $T(n) = O(n^2)$, $T(n) = O(n \log n)$

    Persoalan *searching* → $T(n) = O(n)$, $T(n) = O(\log n)$

    Perkalian matriks → $T(n) = O(n^3)$, $T(n) = O(n^{2.83})$

  - Algoritma yang tergolong "bagus"

- *Nonpolynomial-time algorithm* adalah algoritma yang kempleksitas waktunya dibatasi oleh fungsi non-polinom terhadap ukuran masukannya (n).

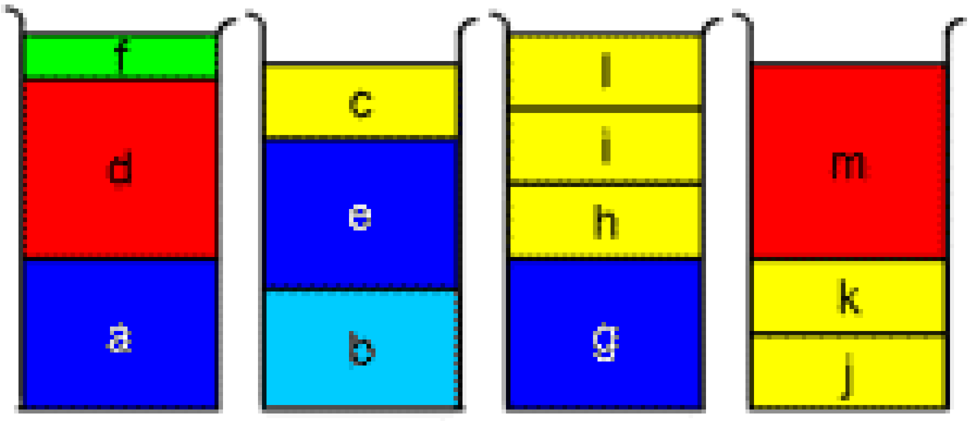
  - Contoh: TSP → $T(n) = O(n!)$

    Integer knapsack problem → $T(n) = O(2^n)$

,   *graph coloring, sum of subset*, *bin packing problem*

  - Persoalan "sulit" (*hard problem*).

- *Bin-packing problem*: Terdapat sejumlah kardus masing-masing dengan kapasitas *C* dan *n* barang berukuran S1, S2, ..., Sn, Kemaslah *n* barang ke dalam *M* kardus sedemikian sehingga ukuran total barang di dalam setiap kardus tidak melebihi C. Temukan minimal *M* yaitu paling sedikit jumlah kardus untuk menampung *n* barang.



Tractable vs *Intractable Problem*

- Sebuah persoalan dikatakan *tractable* jika ia dapat diselesaikan dalam waktu yang wajar (*reasonable*).

- Sebuah persoalan dikatakan *intractable* jika ia tidak dapat diselesaikan dalam waktu yang wajar dengan bertambahkanya ukuran persoalan.

- Apa yang dimaksud dengan waktu yang wajar? Standar waktunya adalah *polynomial time*.
  - *Polynomial time*: $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$
  - *Not in polynomial time*: $O(2^n)$, $O(n^n)$, $O(n!)$ untuk n yang kecil

Solvable vs *Unsolvable Problem*

Dikaitkan dengan Mesin Turing, sebuah persoalan dikatakan:

- *Solvable*, jika terdapat mesin Turing yang dapat menyelesaikannya.

- *Unsolvable*, jika tidak dapat dibuat mesin Turing untuk menyelesaikannya.

- *Solvable problem* dapat dibagi menjadi dua kategori:

  1. *Tractable*
  2. *Intractable*

- Adakah persoalan yang *unsolvable*? Ada, contoh persoalan yang terkenal dikemukakan oleh Alan Turing pada tahun 1963, yaitu *halting problem*.

- *Halting problem*: diberikan sebuah program komputer dan input untuk program tersebut, tentukan apakah program akan berhenti (*halt*) dengan *input* tersebut atau berlanjut bekerja secara tak terbatas (*infinite loop*)?

Alan designed the perfect computer

- Jadi, untuk program *P* dan input *I*,

  A(*P*, *I*) = 1, jika program *P* berhenti untuk masukan *I*
  = 0, jika program *P* tidak berhenti

- Sebagai contoh, kode program berikut

  ```
  while (true) { }
  ```

  akan terus berulang tanpa berhenti (*infinite loop*)

  Sedangkan program

  ```
  printf ("Hello World!“);
  ```

  berhenti dengan sangat cepat.

- Sebuah program yang lebih kompleks mungkin lebih sulit untuk menganalisisnya.
- Masalahnya adalah, apakah program benar-benar berhenti? Bagaimana membuktikan bahwa program benar-benar berhenti?
- Jika program tidak berhenti, tidak ada cara untuk mengetahui apakah program akhirnya akan berhenti atau *loop forever*.
- Turing membuktikan tidak ada algoritma yang dapat diterapkan untuk setiap program dan masukan sembarang untuk memutuskan apakah program berhenti ketika dijalankan dengan masukan itu.

- **Alan Mathison Turing**, (23 June 1912 – 7 June 1954), was an English mathematician, logician, cryptanalyst, and computer scientist. He was highly influential in the development of computer science, providing a formalisation of the concepts of "algorithm" and "computation" with the Turing machine, which played a significant role in the creation of the modern computer. Turing is widely considered to be the father of computer science and artificial intelligence.[3]

Sumber: Wikipedia.org

Algoritma Deterministik

- Algoritma deterministik adalah algoritma yang dapat ditentukan dengan pasti apa saja yang akan dikerjakan selanjutnya oleh algoritma tersebut.

- Algoritma deterministik bekerja sesuai dengan cara program dieksekusi oleh komputer.

- Semua algoritma yang sudah dipelajari sejauh ini adalah algoritma deterministik

Contoh: *Sequential search*

function ***Sequential-Search(A, x)***

{Menghasilkan indeks k sedemikian sehingga A[k]=x atau -1 jika tidak terdapat x di dalam A[1..n] }

Algoritma:

```
k ← 1
while (A[k] ≠ x) and (k < n) do
    k ← k + 1
end
if A[k] = x then
   return k
else
   return -1
end
```

Kompleksitas waktu: O(n)

Algoritma Non-deterministik

- Algoritma non-deterministik adalah algoritma yang mengandung operasi yang berhadapan dengan beberapa opsi pilihan, dan algoritma memiliki kemampuan untuk menerka opsi pilihan yang **tepat**.

- Algoritma non-deterministik dijalankan mesin non-deterministik (komputer hipotetik).

- Meskipun mesin non-deterministik dalam praktek tidak pernah ada, namun konsep mesin non-determinstik memberikan sebuah gagasan untuk menyelesaikan persoalan yang tidak dapat diselesaikan dengan cepat oleh algoritma deterministik.

Ada dua tahap di dalam algoritma non-deterministik:

1. Tahap menerka atau memilih (non-deterministik): Diberikan *instance* persoalan, tahap ini memilih atau menerka satu opsi dari beberapa opsi yang ada. Bagaimana cara membuat pilihan itu tidak didefinisikan aturannya.

2. Tahap verifikasi (deterministik): memeriksa apakah opsi yang diterka menyatakan solusi. Luaran dari tahap ini adalah sinyal **sukses** jika solusi ditemukan atau sinyal **gagal** jika bukan solusi.

Contoh: Non-deterministic Search

Algoritma *Search(A, x)*

{ tahap menerka }

k $\leftarrow$ Pilih(1, n) O(1)

{ tahap verifikasi }

if (A[k] = x) **then** O(1)

write(k); Sukses() O(1)

end

write(-1); Gagal() O(1)

Kompleksitas waktu: O(1) + O(1) + O(1) + O(1) = O(1)

Contoh lain: Sorting

```
Algorithm NSort(A, n)
// Sort n positive integers.
{
    for i := 1 to n do B[i] := 0; // Initialize B[ ].
    for i := 1 to n do
    {
        j := Choice(1, n);
        if B[j] ≠ 0 then Failure();
        B[j] := A[i];
    }
    for i := 1 to n − 1 do  // Verify order.
        if B[i] > B[i + 1] then Failure();
    write (B[1 : n]);
    Success();
}
```

Algorithm 11.2 Nondeterministic sorting

Sumber: Horowitz & Sahni, Fundamental of Computer Algorithms, 2nd Edition

Persoalan Keputusan

- Dalam membahas teori P dan NP, kita hanya membatasi pada persoalan keputusan (*decision problem*)

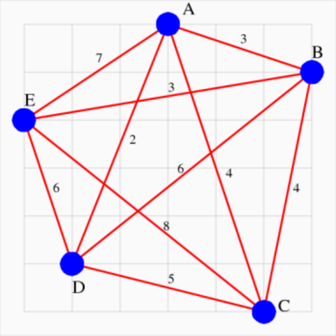
- Persoalan keputusan adalah persoalan yang solusinya hanya jawaban “yes” atau “no”.

  Contoh:
  1. Diberikan sebuah integer x.
     Tentukan apakah elemen *x* terdapat di dalam tabel?
  2. Diberikan sebuah integer x.
     Tentukan apakah x bilangan prima?
  3. Diberikan sebuah graf G, apakah G graf Hamilton?

- Setiap persoalan optimasi yang kita kenal memiliki *decision problem* yang bersesuaian.

- Perhatikan beberapa persoalan berikut:

1. *Travelling Salesperson Problem*

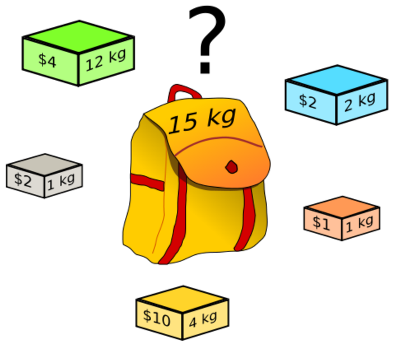


- Diberikan graf berarah dengan bobot (*weight*) pada setiap sisinya. Sebuah tur di dalam graf tersebut dimulai dari sebuah simpul,mengunjungi simpul lainnya tepat sekali dan kembali lagi ke simpul asalnya.

- *Travelling Salesperson Optmization Problem* (TSOP) adalah persoalan menentukan tur dengan total bobot sisi minimal → TSP yang sudah biasa dikenal.

- *Travelling Salesperson Decision Problem* (TSDP) adalah persoalan untuk menentukan apakah terdapat tur dengan total bobot sisinya tidak melebihi nilai *d*.

2. *Knapsack Problem*

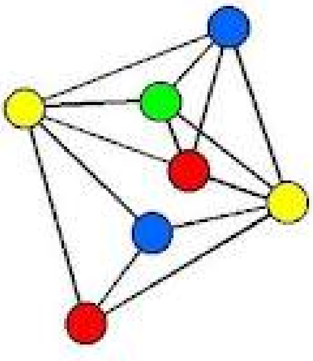


- Diberikan *n* buah objek dan sebuah *knapsack* dengan kapasitas *W*. Setiap objek memiliki profit masing-masing.

- *Integer Knapsack Optimization Problem* adalah menentukan objek-objek yang dimasukkan ke dalam *knapsack* namun tidak melebihi *W* sehingga memberikan total profit maksimum. → *Knapsack problem* yang sudah kita kenal

- *Integer Knapsack Decision Problem* adalah persoalan untuk menentukan apakah dapat memasukkan objek-objek ke dalam *knapsack* namun tidak melebihi *W* tetapi total profitnya paling sedikit sebesar *P*.

3. Graph Colouring Problem



- *Graph-Colouring Optimization Problem* adalah menentukan jumlah minimal warna yang dibutuhkan untuk mewarnai graf sehingga dua simpul bertetangga memiliki warna berbeda. → *Graph Colouring problem* yang kita kenal.

- *Graph-Colouring Decision Problem* adalah menentukan, untuk suatu integer *m*, apakah terdapat pewarnaan yang menggunakan paling banyak *m* warna sedemikian sehingga dua simpul bertetangga memiliki warna berbeda.

- Kita belum menemukan algoritma polinomial untuk persoalan optimasi atau persoalan keputusan pada contoh-contoh di atas.
- Namun, jika kita dapat menemukan algoritma polinomial untuk jenis persoalan optimasi tersebut, maka kita juga mempunyai algoritma waktu-polinom untuk persoalan keputusan yang bersesuaian.
- Hal ini karena solusi persoalan optimasi menghasilkan solusi persoalan keputusan yang bersesuaian.

- Contoh: jika pada persoalan *Travelling Salesperson Optimization Problem* (TSOP) tur minimal adalah 120,

- maka jawaban untuk persoalan *Travelling Salesperson Decision Problem* (TSDP) adalah “yes” jika $d \leq 120$, dan “no” jika $d > 120$.

- Begitu juga pada persoalan *Integer Knapsack Optimization Problem*, jika keuntungan optimalnya adalah 230, jawaban untuk persoalan keputusan *integer knapsack* yang berkoresponden adalah “yes” jika $P \geq 230$, dan “no” jika $P < 230$.

Dua tahap di dalam algoritma non-deterministik untuk persoalan keputusan:

1. Tahap menerka (non-determinsitik): Diberikan *instance* persoalan, tahap ini secara sederhana (misalnya) menghasilkan beberapa string *S*. String ini dapat dianggap sebagai sebuah terkaan pada sebuah solusi. String yang dihasilkan bisa saja tidak bermakna (*non-sense).*

2. Tahap verifikasi (deterministik): memeriksa apakah S menyatakan solusi. Luaran tahap ini adalah “true” jika *S* merupakan solusi, atau “false” jika bukan.

Agoritma non-deterministik TSDP:

- Tahap menerka

  S $\leftarrow$Terka(string)

- Tahap verifikasi

  Solusi dapat diverifikasi dengan menghitung semua bobot sisi yang terpilih dan memeriksa apakah jumlahnya lebih kecil dari *d*

  if S adalah tur dan total bobot $\leq$ d **then**

  return **true**

  else

  return **false**

  end

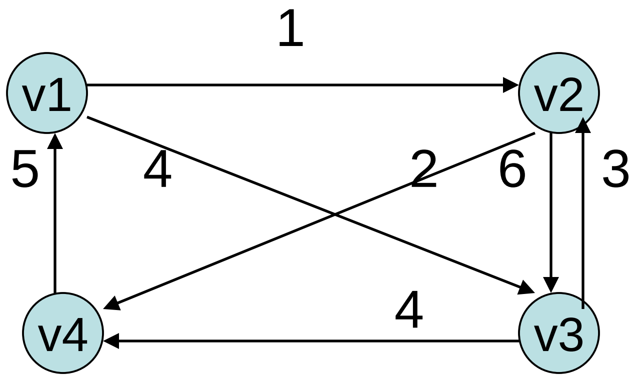
Kompleksitas waktu: O(n)

- Algoritma non-deterministik dikatakan “menyelesaikan” (*completion*) persoalan keputusan apabila:

  1. Untuk suatu *instance* dimana jawabanya adalah “yes”, terdapat beberapa string S yang pada tahap verifikasi menghasilkan “true”

  2. Untuk suatu *instance* dimana jawabannya adalah “no”, tidak terdapat string S yang pada tahap verifikasi menghasilkan “true”.

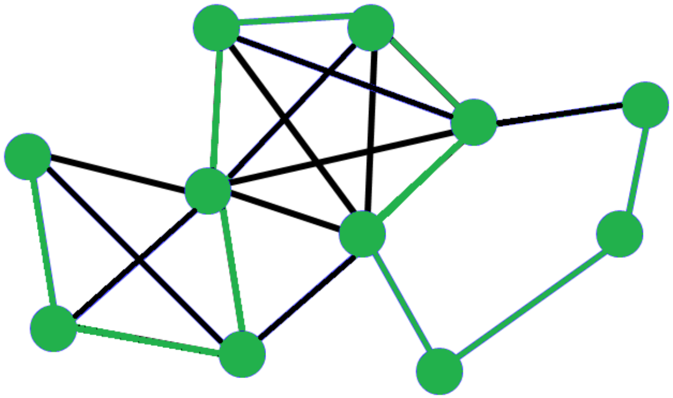
- Contoh untuk TSDP dengan d = 15:



S	Keluaran	Alasan
[v1, v2, v3, v4, v1]	False	Total bobot > 15
[v1, v4, v2, v3, v1]	False	S bukan sebuah tur
^%@12*&a%	False	S bukan sebuah tur
[v1, v3, v2, v4, v1]	True	S sebuah tur, total bobot $\leq$ 15

- Kita dapat menyatakan bahwa algoritma non-deterministik "menyelesaikan" TSDP dalam dua tahap tersebut

- **Persoalan sirkuit Hamilton**: Diberikan sebuah graf G. Apakah G mengandung sirkuit Hamilton? Sirkuit Hamilton adalah sirkuit yang melalui setiap simpul di dalam graf tepat satu



Algoritma non-deterministik:

1. Terkalah permutasi semua simpul
2. Verifikasi: Periksa apakah permutasi tersebut membentuk sirkuit. Jika ya, maka itulah solusinya. STOP.


I helped solve
The
Travelling
Salesman
problem

P Problems

- *P Problems* adalah himpunan semua persoalan keputusan yang dapat dipecahkan oleh algoritma dengan kebutuhan waktu polinom.

- Semua persoalan keputusan yang dapat diselesakan dalam waktu polinom adalah anggota himpunan *P*.

  Contoh: Persoalan mencari sebuah nilai di dalam sebuah larik adalah *P Problems*.

- Semua persoalan keputusan yang telah ditemukan algoritma dalam waktu polinom termasuk ke dalam *P Problems*.

- Apakah *Travelling Salesperson Decision Problem* (TSDP) termasuk *P Problems*?

- Meskipun belum ada orang yang menemukan algoritma TSDP dalam waktu polinom, namun tidak seorang pun dapat membuktikan bahwa TSDP tidak dapat dipecahkan dalam waktu polinom.

- Ini berarti TSDP *mungkin* dapat dimasukkan ke dalam *P Problems*.

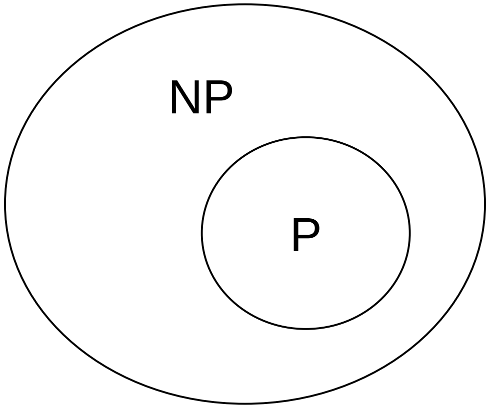
NP Problems

- *NP = non-deterministic polynomial*
- *Non-deterministic polynomial-time algorithm* adalah algoritma non-deterministik dimana tahap verifikasinya dapat dilakukan dalam waktu polinomial.
- *NP Problems* adalah himpunan persoalan keputusan yang dapat diselesaikan oleh algoritma non-deterministik dalam waktu polinom.
- Kebanyakan persoalan keputusan adalah NP

- TSDP adalah contoh persoalan NP, sebab jika diberikan sebuah terkaan string (tur), maka dibutuhkan O(n) untuk memverifikasi solusi.

- *Integer Knapsack Decision problem* dan *Graph Coloring Decision Problem* semuanya adalah NP.

- Karena algoritma deterministik adalah kasus khusus dari algoritma non-determinsitik, maka kita menyimpulkan $P \subseteq NP$. Alasannya, tahap menerka tidak terdapat di dalam persoalan P.



$$P \subseteq NP$$

- $P \subseteq NP$ mengindikasikan dua hal:
  (i) $P = NP$ atau (ii) $P \neq NP$
- Tidak seorangpun pernah membuktikan bahwa $P \neq NP$ atau $P = NP$.
- Pertanyaan apakah $P = NP$ adalah salah satu pertanyaan penting dalam ilmu komputer.
- Pertanyaan ini sangat penting sebab, seperti telah disebutkan sebelumnya, kebanyakan persoalan keputusan adalah NP.

- Karena itu, jika P = NP, maka betapa banyak persoalan keputusan yang dapat dipecahkan secara mangkus dengan algoritma yang kebutuhan waktunya polinom.
- Namun kenyataannya, banyak ahli yang telah gagal menemukan algoritma waktu-polinom untuk persoalan NP.
- Karena itu, cukup aman kalau kita mengasumsikan bahwa P $\neq$ NP

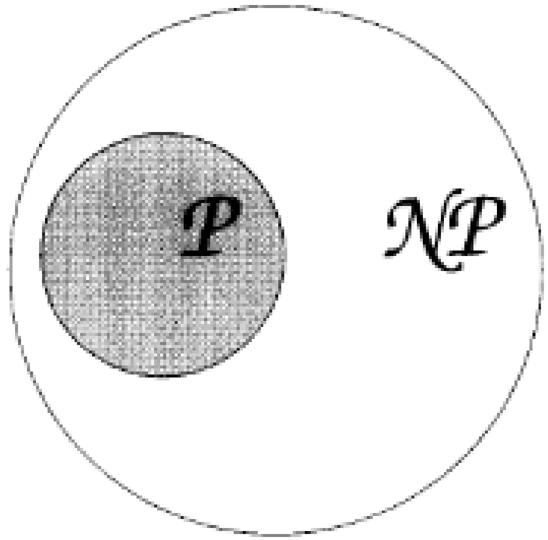


Figure 11.1 Commonly believed relationship between $\mathcal{P}$ and $\mathcal{NP}$

- Adakah persoalan yang tidak termasuk ke dalam NP? Ada, yaitu persoalan *unsolvale*. Contohnya *halting problem*.

- The **P versus NP problem** is a major unsolved problem in computer science. Informally, it asks whether every problem whose solution can be quickly verified by a computer can also be quickly solved by a computer. It was introduced in 1971 by Stephen Cook in his seminal paper "The complexity of theorem proving procedures"[2] and is considered by many to be the most important open problem in the field.[3] It is one of the seven Millennium Prize Problems selected by the Clay Mathematics Institute to carry a US$1,000,000 prize for the first correct solution.

(Sumber: Wikipedia)

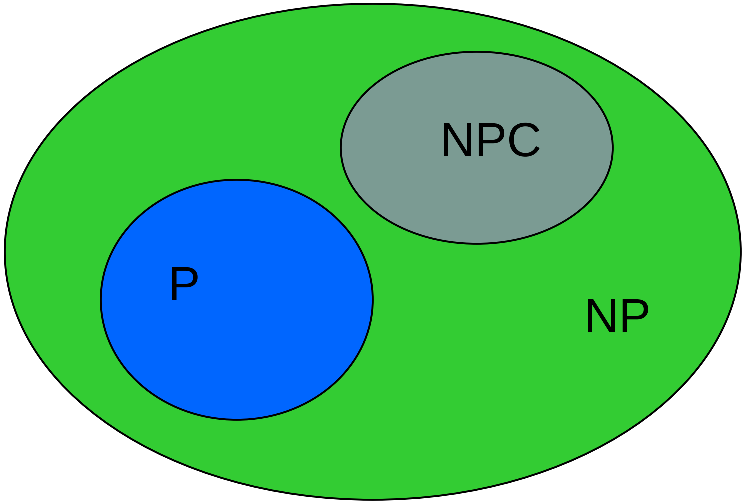
The $1M question

The Clay Mathematics Institute Millenium Prize Problems:

1. Birch and Swinnerton-Dyer Conjecture
2. Hodge Conjecture
3. Navier-Stokes Equations
4. P vs NP
5. Poincaré Conjecture
6. Riemann Hypothesis
7. Yang-Mills Theory

NP-Complete Problems

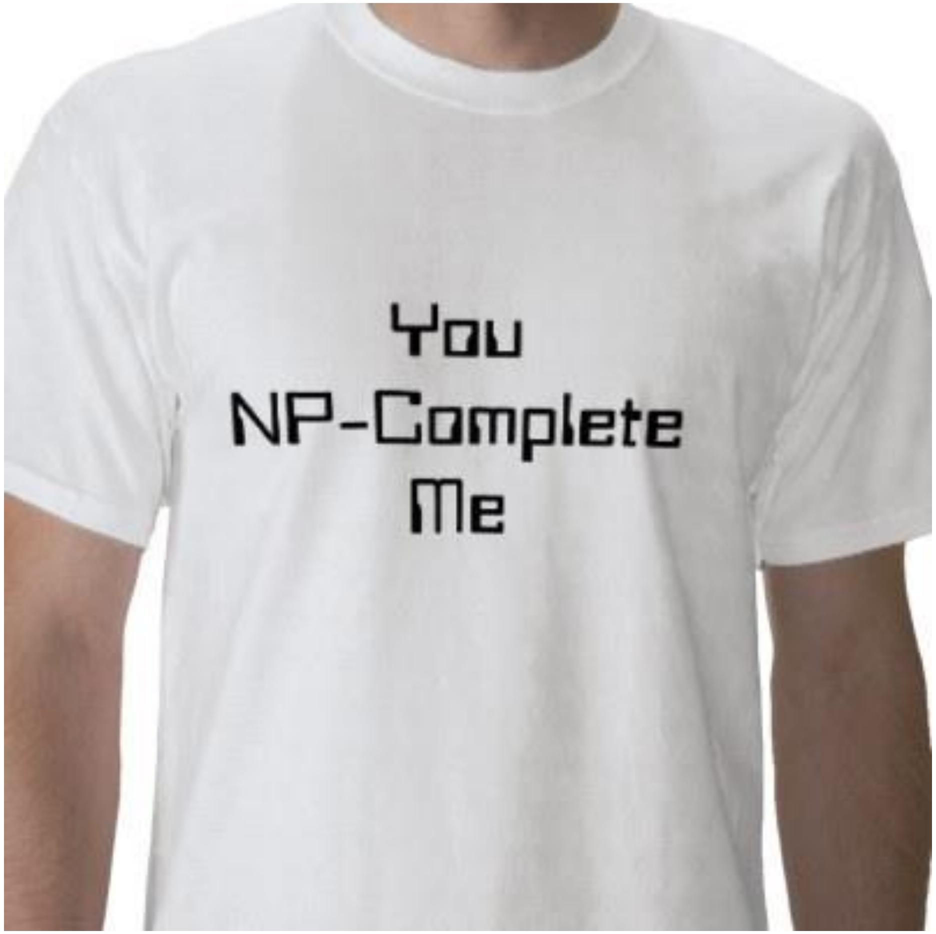
- NP-Complete (NPC) adalah persoalan NP yang paling menarik perhatian.



- **Definisi NPC.** Sebuah persoalan X dikatakan NPC jika:

  1) X termasuk ke dalam kelas NP

  2) Setiap persoalan di dalam NP dapat direduksi menjadi X dalam waktu polinom


You
NP-Complete
Me

- Cara termudah untuk membuktikan sebuah persoalan X adalah NPC adalah menemukan sebuah metode sederhana (algoritma dalam waktu polinom) untuk mentransformasikan persoalan yang sudah dikenal NPC menjadi persoalan X tersebut.

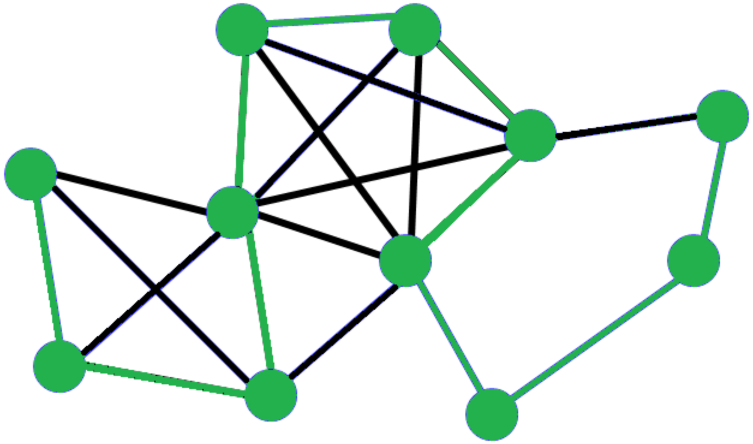
- Dengan kata lain, untuk menunjukkan bahwa X adalah NPC, caranya adalah sebagai berikut:
  1) Tunjukkan bahwa X adalah anggota NP
  2) Pilih *instance,* Y, dari sembarang persoalan NPC.
  3) Tunjukkan sebuah algoritma dalam waktu polinom untuk mentransformasikan Y menjadi *instance* persoalan X

FYI, nama "NP-Complete" berasal dari:

- **Nondeterministic Polynomial**
- **Complete - "Solve one, Solve them all"**

- **Contoh**: TSP adalah persoalan yang sudah dikenal NPC. Kita ingin menunjukkan persoalan sirkuit Hamilton (HCP, *Hamiltonian Circuit Problem*) termasuk ke dalam NPC. Kita pilih TSP untuk menunjukkan HCP termasuk ke dalam NPC.

  Persoalan HCP: Diberikan sebuah graf *G* dengan *n* buah simpul, tentukan apakah graf tersebut mengandung sirkuit Hamilton. Sirkuit Hamilton adalah sirkuit yang melalui setiap simpul di dalam graf *G*.



Perhatikan bahwa sirkuit Hamilton di dalam graf G dengan n simpul akan mempunyai n buah sisi

- Untuk mentransformasikan instans HCP menjadi instans TSP, maka algoritma transformasi yang sederhana adalah sbb:

  1. Setiap sisi di dalam graf G diberi nilai (bobot) 1
  2. Nyatakan persoalan menjadi TSP, yaitu adakah tur dengan total bobot $\leq n$?

- Dengan transformasi ini, maka persoalan HCP sudah ditransformasi menjadi instans persoalan TSP.

- Transformasi ini (yaitu memberi bobot setiap sisi dengan nilai 1) membutuhkan waktu polinom, yaitu O(m), m adalah jumlah sisi di dalam graf.

- Misalkan di dalam graf G = (V, E), |E| = m, yaitu jumlah sisi di dalam graf adalah n. Maka, algoritma memberi setiap sisi di dalam graf G dengan 1 adalah sbb:

for tiap sisi (u, v) $\in$ E **do**
  (u, v) $\leftarrow$ 1
end

Jumlah pengulangan untuk (u, v) $\leftarrow$ 1 adalah sebanyak m kali, sehingga: $T(n) = m = O(m)$ → polinomial

- Transformasi ini memberi sugesti bahwa jika TSP dapat diselesaikan dengan mangkus (kebutuhan waktu dalam polinom), maka HCP juga dapat diselesaikan dengan mangkus.

- Tinjau kembali definisi NPC. Sebuah persoalan X dikatakan NPC jika:

  1. X termasuk ke dalam kelas NP

  2. Setiap persoalan di dalam NP dapat direduksi dalam waktu polinom menjadi X

- Definisi ini menyatakan jika transformasi dari sembarang persoalan NP menjadi instans persoalan NPC dapat dilakukan, maka jika algoritma dalam waktu polinom ditemukan untuk X, maka semua persoalan di dalam NP dapat diselesaikan dengan mangkus.

- Dengan kata lain, jika X adalah NPC dan termasuk ke dalam P – yaitu algoritma dalam waktu polinom untuk X ditemukan -- maka dapat menjawab bahwa P = NP.

- Persoalan di dalam NPC dikatakan persoalan yang paling sukar (*hardest*) karena jika ada persoalan NPC dipecahkan dalam waktu polinomial, maka semua persoalan di dalam NP dapat dipecahkan dalam waktu polinomial.

- Sebaliknya, jika P $\neq$ NP, maka tidak ada persoalan NPC dapat dipecahkan dalam waktu polinomial. Sebagai konsekuensinya, jika satu persoalan NP *intractable*, maka semua persoalan NPC adalah *intractable*. Inilah alasan lain kenapa NPC dipandang sebagai *the hardest problem*.

> If any single *NP-complete* problem can be solved in polynomial time, then *all* problems in *NP* can be solved in polynomial time.
>
> If any single problem in *NP* is intractable, then *all* *NP-complete* problems are intractable

- Sejauh ini, lebih dari 300 persoalan yang sudah terbukti NP-complete

- Persoalan NP-Complete lainnya:
  - PARTITION problem
  - SUM OF SUBSET problem
  - CLIQUE problem
  - GRAPH COLORING problem
  - SAT (Satisfiability problem)
  - Vertex cover
  - N-PUZZLE
  - Knapsack problem
  - Subgraph isomorphism problem
  - MINESWEEPER

- PARTITION: Diberikan *n* buah bilangan bulat positif. Bagilah menjadi dua himpunan bagian *disjoint* sehingga setiap bagian mempunyai jumlah nilai yang sama (catatan: masalah ini tidak selalu mempunyai solusi).

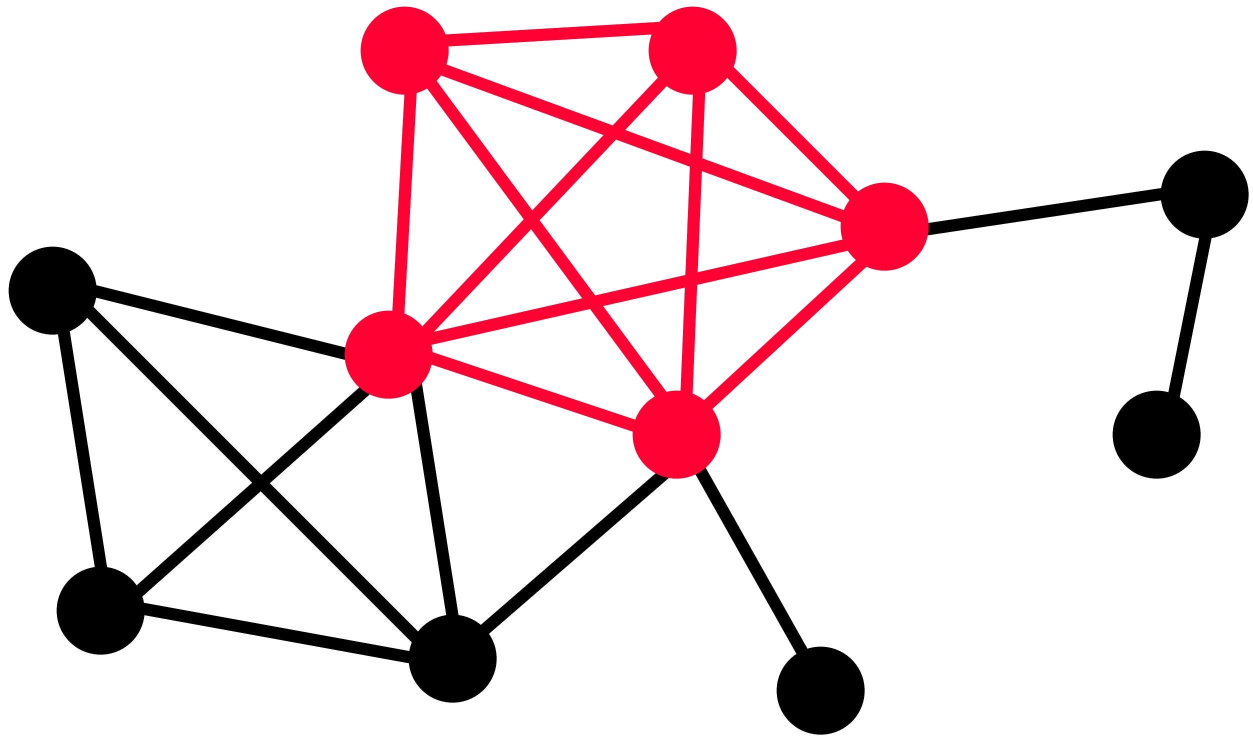
  Contoh: *n* = 6, yaitu 3, 8, 4, 6, 1, 2, dibagidua menjadi {3, 8, 1} dan {4, 6, 2} yang masing-masing jumlahnya 12.

- SUM OF SUBSET: Diberikan sebuah himpunan bilangan bulat. Carilah upahimpunan yang jumlahnya = *m*.

  Contoh: A = {-4, -1, 1, 2, 3, 8, 9} dan *m* = 0.

  Maka salah satu solusinya adalah {-4, -1, 2, 3}

- CLIQUE: sebuah *clique* adalah subset dari himpunan simpul di dalam graf yang semuanya terhubung



Upagraf yang berwarna merah adalah sebuah *clique*

Sumber: *Complexity Theory, based on*

Garey M., Johnson D.S., *Computers and Intractability A guide to the Theory of NP-Completeness,* Freeman and Company - New York - 2000

Sumber: *Complexity Theory, based on* Garey M., Johnson D.S., *Computers and Intractability A guide to the Theory of NP-Completeness,* Freeman and Company - New York - 2000

SAT

- SAT = *Satisfiability Problem*

Up to now we never encountered *NP*-complete problems

The first example of NP-complete problem was found by Cook in 1971

(before this date, the concept of *NP*-completeness did not even exist)

Given $X = \{x_1, x_2, \ldots, x_n\}$ a set of Boolean variable, that can assume value $\{0,1\}$, and a *clause* over X, that is a set containing variables or negation of variables, a collection C of clauses is *satisfiable* if and only if there exists some truth assignment for X that simultaneously satifies all the clauses.

SATISFIABILITY PROBLEM (SAT) in Boolean algebra

Instance: a set X of variables and a collection C of clauses
Question: is there a satisfying truth assignment for C?

Example *Yes*

$X = \{x_1, x_2, x_3\}$ $\qquad$ $C = (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (x_2) \wedge (x_1 \vee \bar{x}_2)$

The truth assignment $x_1 = 1, x_2 = 1, x_3 = 1$ satisfies C.

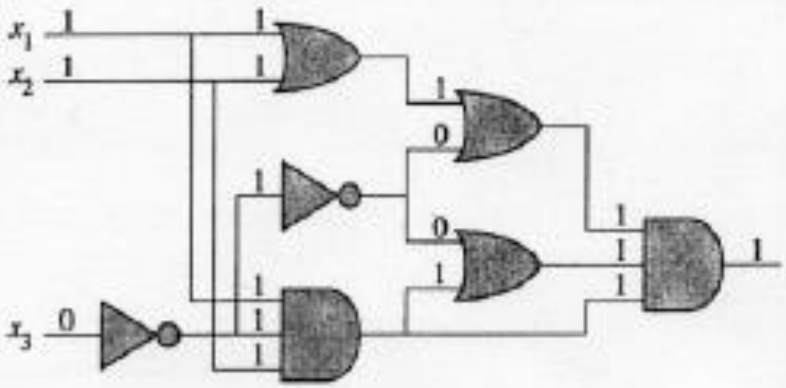
The answer is *Yes*

Example *No*

$X = \{x_1, x_2, x_3\}$ $C' = (x_1 \vee x_2) \wedge (x_1 \vee \overline{x}_2) \wedge \overline{x}_1 \wedge (x_1 \vee x_3)$

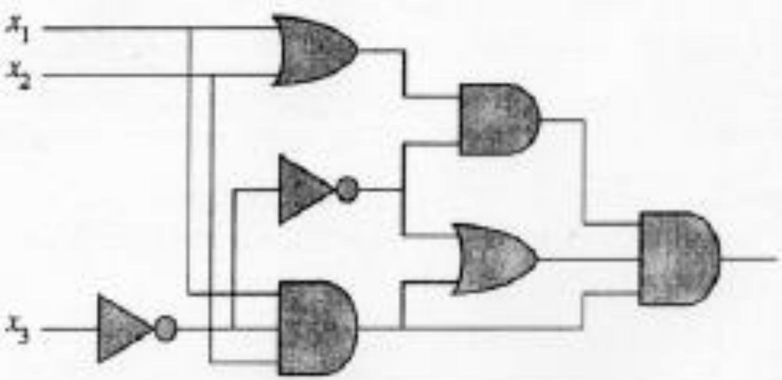
There are no truth assignments that satisfies C'

The answer is *No*



(a)

satisfiable



not satisfiable

Cook's Theorem (1971)

SATISFIABILITY PROBLEM (SAT) in Boolean algebra is *NP*-complete

proof: very complex, since there are infinitely many languages in *NP*, and we cannot prove directly that, for each $L \in NP$ we have $L \propto L_{SAT}$, showing a transformation for each language. We prove the theorem in two steps:

1) SAT is in NP because any assignment of Boolean values to Boolean variables that is claimed to satisfy the given expression can be *verified* in polynomial time by a deterministic Turing machine.

2) Now suppose that a given problem in NP can be solved by the nondeterministic Turing machine NDTM. Suppose further that NDTM accepts or rejects an instance I of the problem in time $p(n)$.
For each input, I, we specify a Boolean expression which is satisfiable if and only if the machine NDTM accepts I.

T A M A T