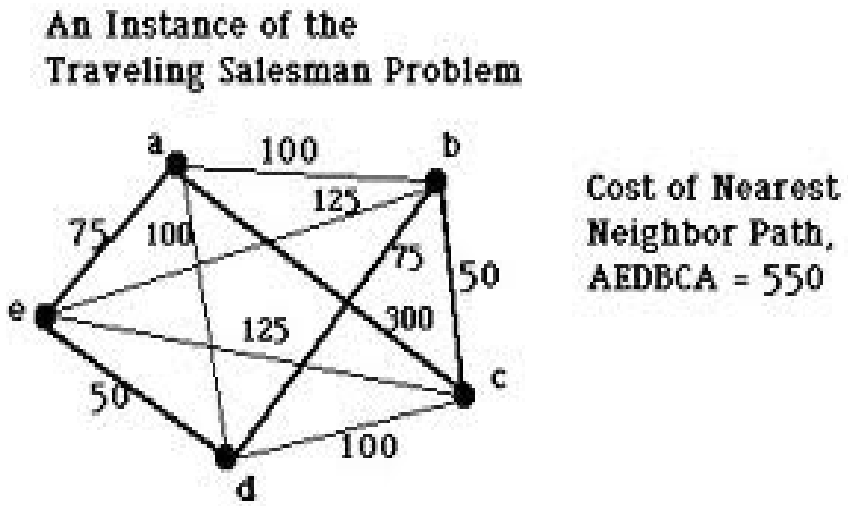
IF5110 Teori Komputasi

Teori Kompleksitas
(Bagian 2)

Oleh: Rinaldi Munir

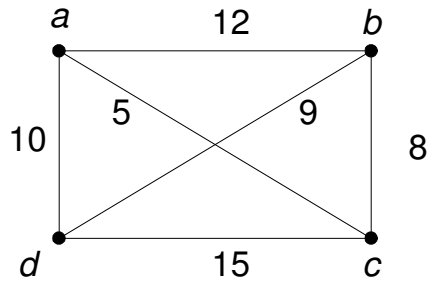
Travelling Salesperson Problem

- Persoalan optimasi.

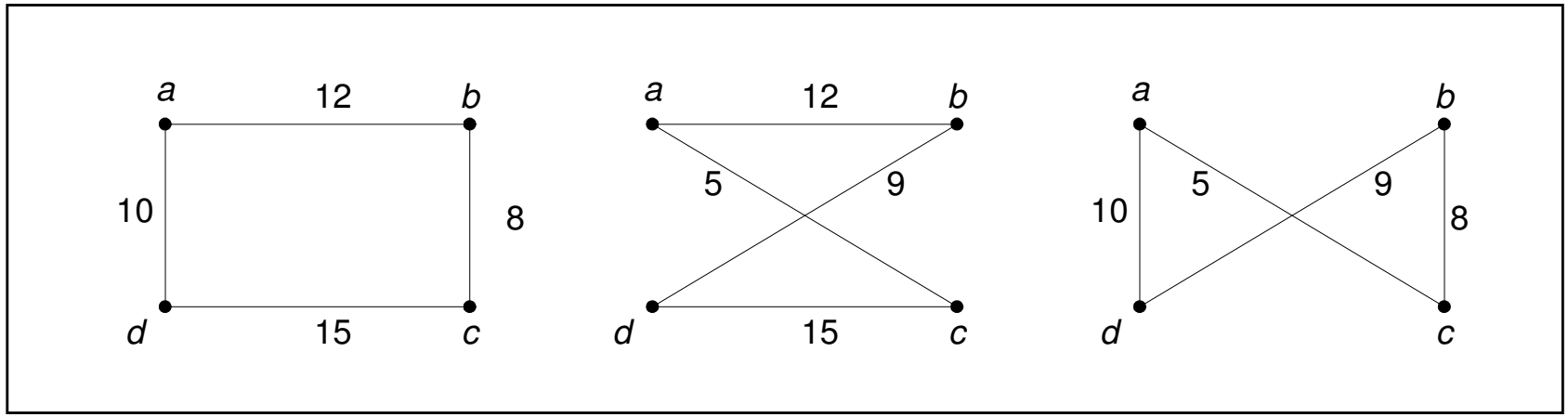


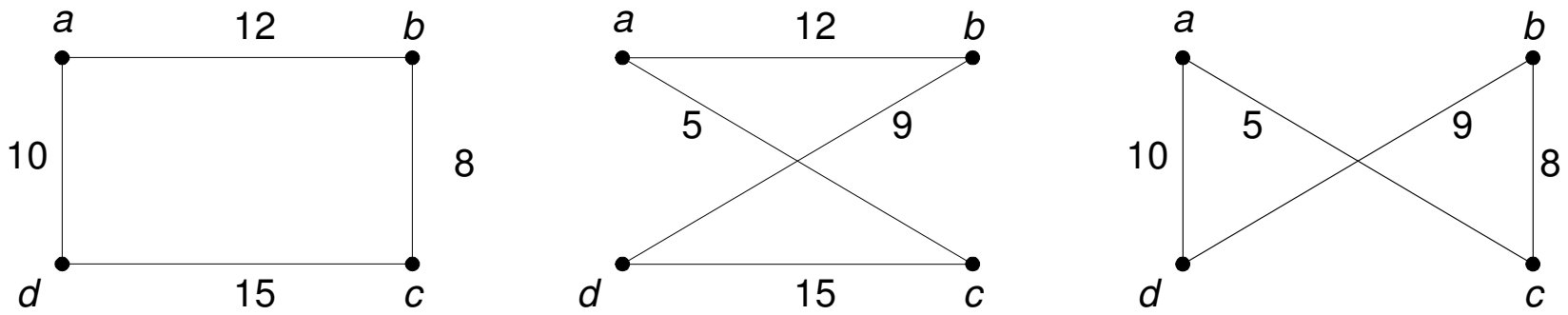
- Termasuk ke dalam kelas persoalan NP

Jumlah sirkuit Hamilton di dalam graf lengkap dengan n simpul: $(n-1)!/2 = O(n!)$.



Graf di atas memiliki $(4-1)!/2 = 3$ sirkuit Hamilton, yaitu:





I_1 = (*a*, *b*, *c*, *d*, *a*) → bobot = 10 + 12 + 8 + 15 = 45

I_2 = (*a*, *c*, *d*, *b*, *a*) → bobot = 12 + 5 + 9 + 15 = 41

I_3 = (*a*, *c*, *b*, *d*, a) → bobot = 10 + 5 + 9 + 8 = 32

Sirkuit Hamilton terpendek: I_3 = (*a*, *c*, *b*, *d*, *a*)
dengan bobot = 10 + 5 + 9 + 8 = 32.

- Jika jumlah simpul *n* akan terdapat (n!)/2 sirkuit Hamilton.
- Jadi, kompleksitas waktunya O(*n*!).

- Persoalan keputusan yang bersesuaian: *Travelling Salesperson Decision Problem* (*TSDP*): Apakah sebuah graf berbobot memiliki tur terpendek yang melalui setiap simpul tepat sekali dan kembali ke simpul awal dengan bobot ≤ *M*?

- Untuk selanjutnya, jika disebutkan TSP maka yan g dimaksudkan adalah TSDP.

- Pada graf di atas, jika $M = 32$ atau lebih, maka jawaban persoalan TSP adalah “yes”, dan jika $M < 32$ maka jawabannya adalah “no”.

Penyelesaian TSP dengan Mesin Turing Deterministik

- Jika diselesaikan dengan mesin Turing deterministik (DTM), kompleksitas algoritmanya eksponensial
  - Periksa setiap sirkuit Hamilton. Jika jumlah bobotnya ≤ *M*, maka jawabannya "yes". Jika tidak, periksa lagi sirkuit Hamilton berikutnya
  - Ada *n*! sirkuit Hamilton yang harus diperiksa, sehingga kompleksitas waktu TSP adalah O(*n*!)

Penyelesaian TSP dengan Mesin Turing Non-deterministik

- Jika diselesaikan dengan mesin Turing non-deterministik (NTM), maka langkah-langkahnya adalah:
  - Terkalah permutasi simpul-simpul, lalu hitung total bobot tur dengan urutan simpul-simpul tersebut.
  - Pada mesin Turing nondeterministik yang *multitape*, menerka permutasi memerlukan $O(n^2)$ gerakan, dan menghitung total bobot memerlukan $O(n^2)$ gerakan, sehingga kompleksitasnya $O(n^2) + O(n^2) = O(n^2)$.

- Ada teorema sbb:

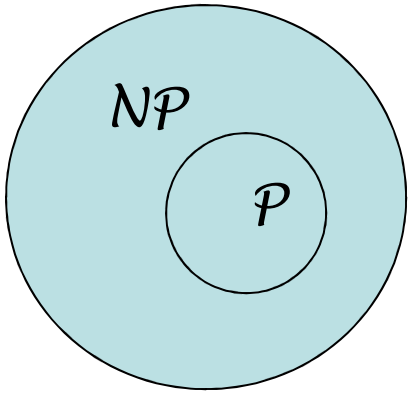
Teorema: Waktu yang dibutuhkan oleh mesin Turing pita tunggal (*one-tape*) untuk mensimulasikan mesin Turing *multi-tape* adalah $O(n^2)$.

- Jadi, jika mesin Turing non-deterministik *multi-tape* menyelesaikan persoalan TSP dalam $O(n^2)$ gerakan, maka mesin Turing non-deterministik pita tunggal menyelesaikan TSP dalam waktu $O((n^2)^2) = O(n^4)$ gerakan.

- Karena mesin Turing non-deterministik menyelesaikan TSP dalam waktu polinomial, maka TSP adalah NP.

- Apakah TSP termasuk kelas NP?
  - Ya!

- Apakah TSP termasuk kelas P?
  - Kita tidak dapat menunjukkan algoritma dengan kompleksitas waktu polinomial pada mesin Turing deterministik.
  - Kita tidak dapat membuktikan bahwa algoritma polinomial tersebut tidak ada.

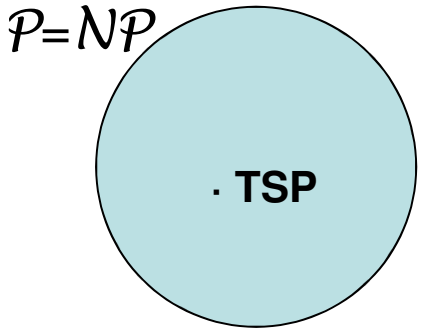


Sumber: Juan Carlos Guzmán
CS 6413 Theory of Computation
Southern Polytechnic State University

Diagram Venn manakah yang akurat untuk TSP?

(i) $\mathcal{P}=\mathcal{NP}$

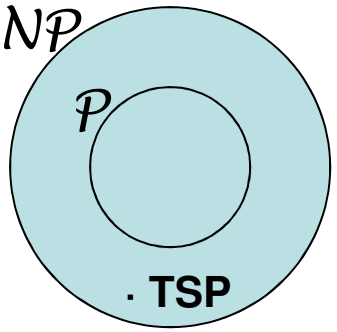
(a)



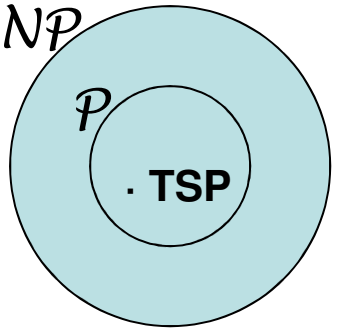
Sumber: Juan Carlos Guzmán
CS 6413 Theory of Computation
Southern Polytechnic State University

(ii) $\mathcal{P}\neq\mathcal{NP}$

(b)



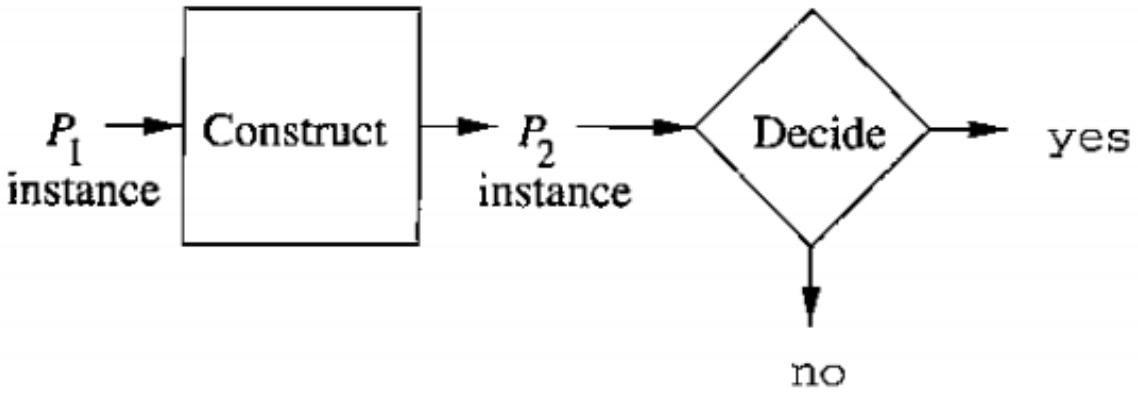
(c)



Reduksi dalam Waktu Polinom

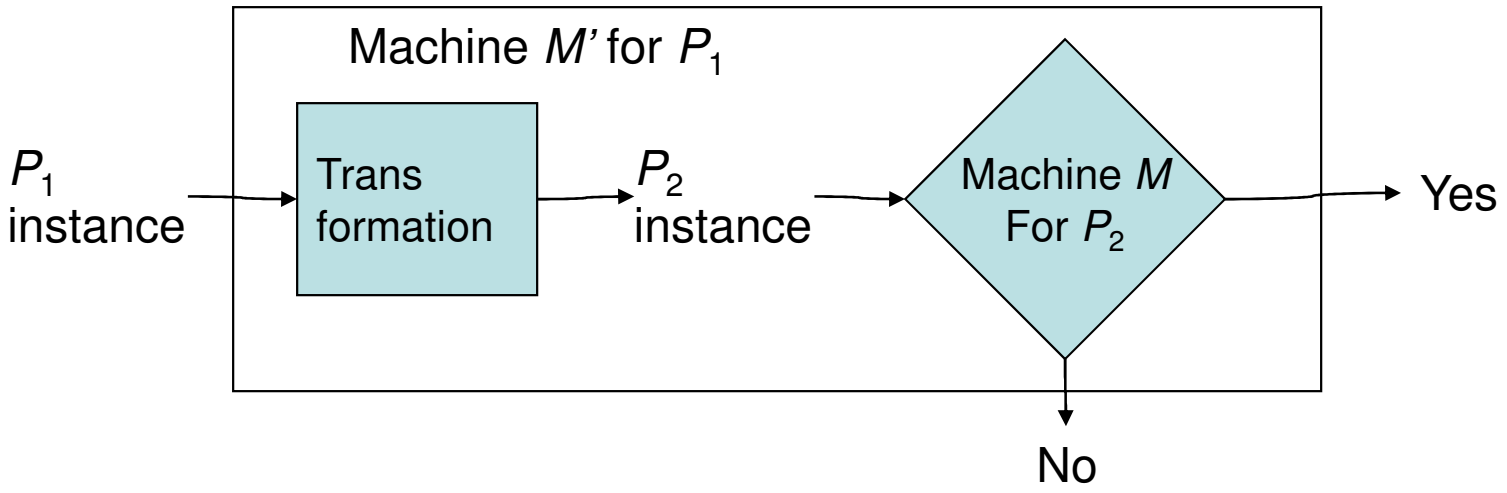
- Ingat kembali diagram reduksi pada materi *undecidability:*

Misalkan P_1 diketahui *undecidable*, dan kita ingin membuktikan bahwa sebuah persoalan baru, P_2, *undecidable*.

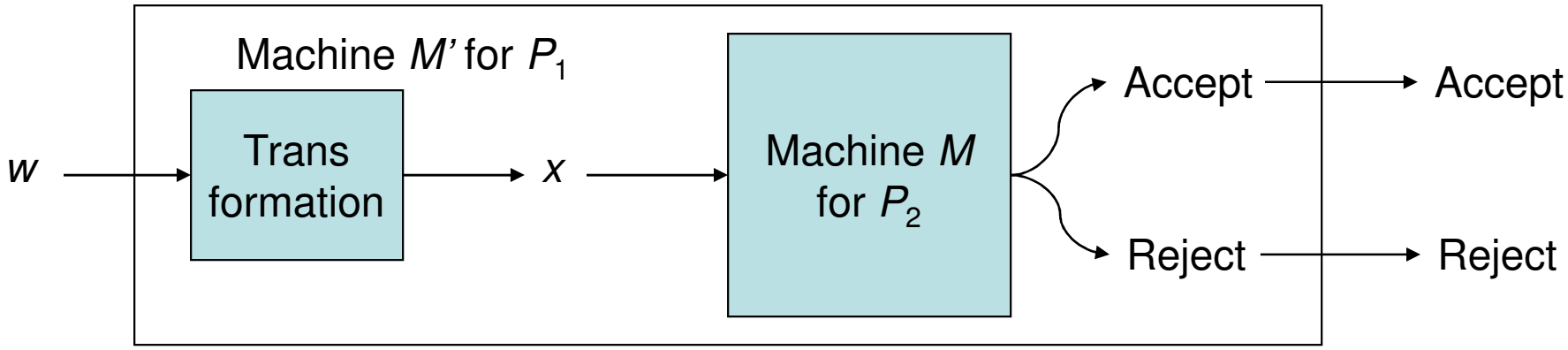


- Ide reduksi ini akan digunakan untuk menunjukkan P_2 tidak dapat dipecahkan dalam waktu polinomial (yaitu P_2 tidak termasuk kelas P).

- Caranya adalah dengan mereduksi instans P_1 (yang diketahui tidak termasuk kelas P) menjadi instans P_2.



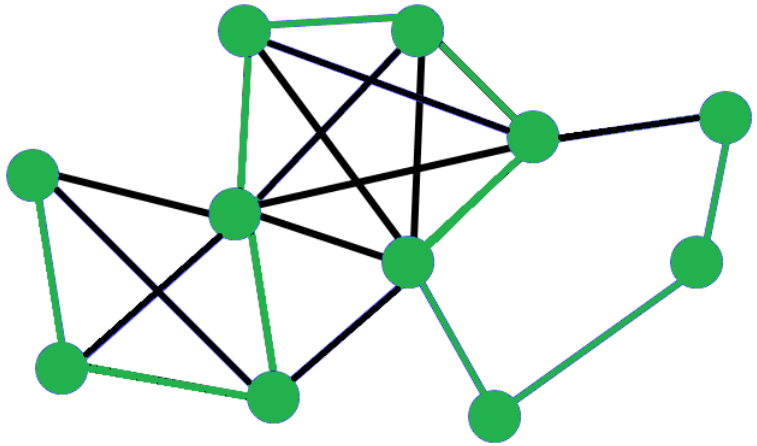
Sumber: Juan Carlos Guzmán
CS 6413 Theory of Computation Southern Polytechnic State University



- Reduksi dari instans P_1 menjadi instans P_2 adalah polinomial.
- Artinya, algoritma reduksi dari P_1 menjadi P_2 kebutuhan waktunya polinomial.

- **Contoh**: Kita ingin menunjukkan transformasi dari Persoalan Sirkuit Hamilton (HCP, *Hamiltonian Circuit Problem*) ke persoalan TSP adalah polinomial.

  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 dalam *term* |E|, 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:

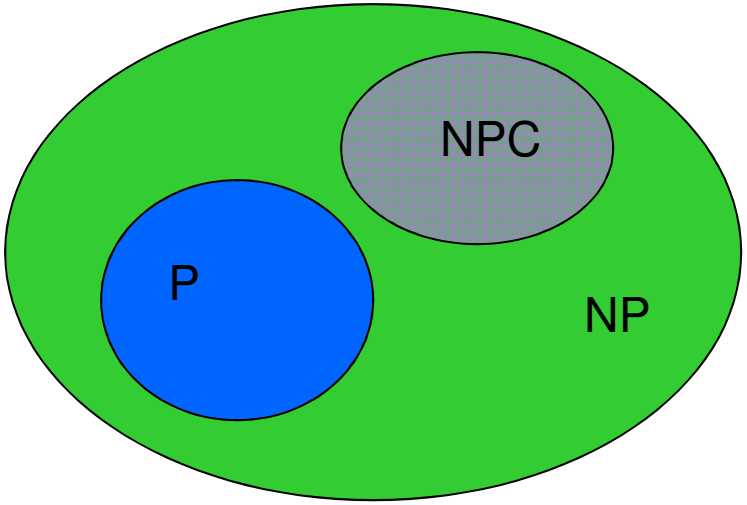
for tiap sisi $(u, v) \in E$ **do**
    $(u, v) \leftarrow 1$
end

Jumlah pengulangan untuk $(u, v) \leftarrow 1$ adalah sebanyakan m kali, sehingga:

$T(n) = m = O(m)$ → polinomial

NP-Complete Problems

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



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

  1) X termasuk ke dalam kelas NP

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

- Dengan kata lain, persoalan X adalah jenis persoalan khusus di dalam NP tetapi tidak di dalam P.

- Karena sebuah persoalan berkoresponden dengan bahasa, maka kita katakan bahasa *L* adalah *NP-complete* jika:

  1) *L* termasuk di dalam NP

  2) Setiap bahasa *L'* di NP dapat direduksi dalam waktu polinomial menjadi *L*.

- Teorema: Jika P_1 *NP-Complete*, P_2 di dalam NP, dan tedapat reduksi dalam waktu polinomial dari P_1 ke P_2, maka P_2 adalah *NP-Complete*.

- Contoh persoalan yang sudah diketahui NPC adalah *Travelling Salesperson Problem* (TSP).

- Cara termudah untuk membuktikan sebuah persoalan X adalah NPC adalah menemukan sebuah metode reduksi sederhana (algoritma dalam waktu polinom) untuk mentransformasikan persoalan yang sudah diketahui 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 (mereduksi) Y menjadi *instance* persoalan X

FYI, nama "NP-Complete" berasal dari:

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

- 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 yang mangkus (polinom, deterministik) untuk X ditemukan -- maka dapat menjawab bahwa P = NP. Hal ini karena transformasi tersebut sederhana membutuhkan waktu polinom.

- **Teorema**: Jika beberapa persoalan NP-Complete termasuk di dalam P, maka 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*.

- Contoh Persoalan *NP-Complete* lainnya:
  - SAT
  - PARTITION problem
  - SUM OF SUBSET problem
  - CLIQUE problem
  - GRAPH COLORING 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).

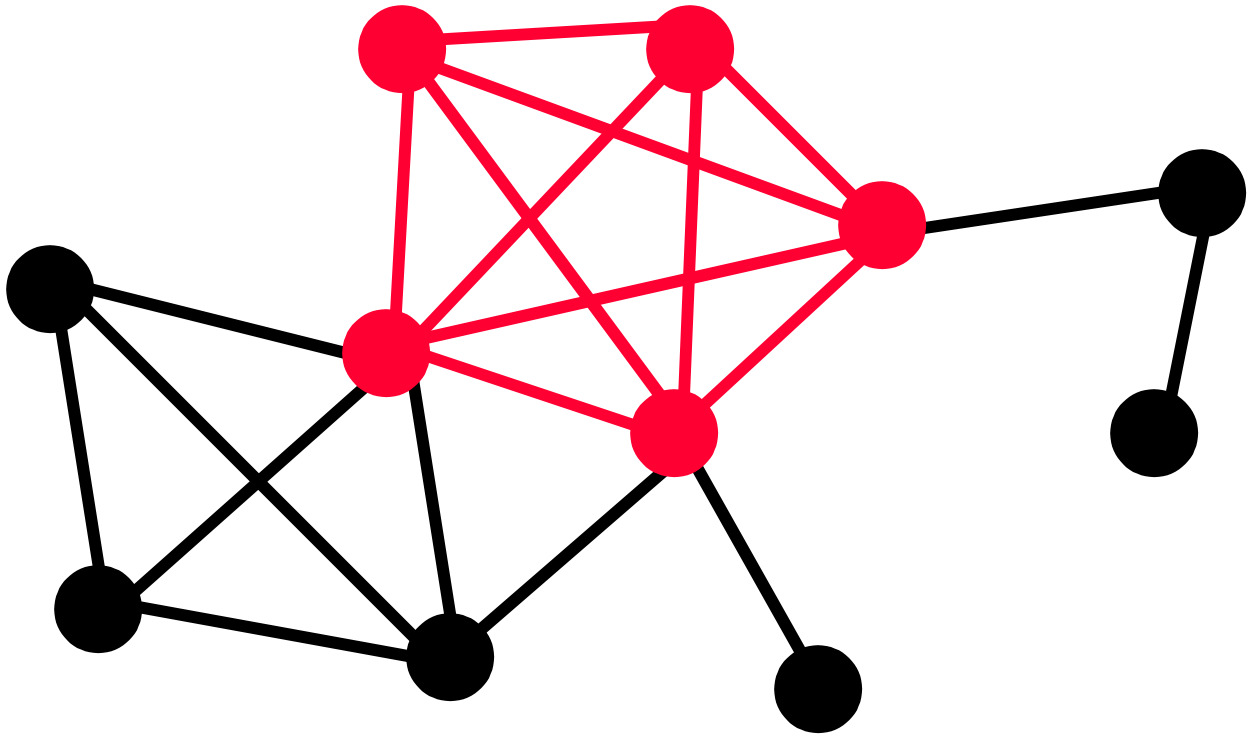
  Contoh: $n = 6$, yaitu 3, 8, 4, 6, 1, 2, dibagidua menjadi {3, 8, 1} dan {4, 6, 2} yang masing-masing jumlahnya 12.

- 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*