13 - Teori P, NP, NP-Complete

[KOMS124404]

Desain dan Analisis Algoritma (2024/2025)

Dewi Sintiari

Prodi S1 Ilmu Komputer
Universitas Pendidikan Ganesha

Week 13 (Mei 2025)

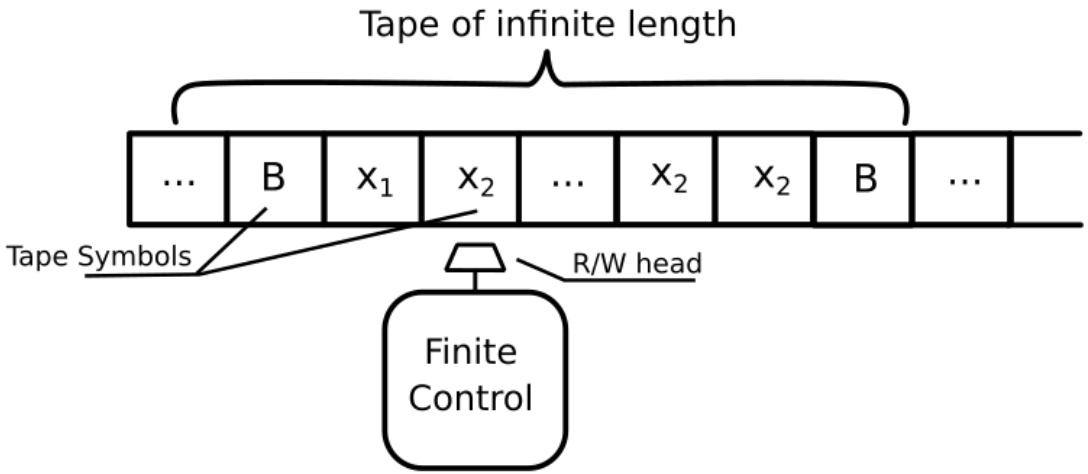
Daftar isi

- Mesin Turing
- Permasalahan P
- Permasalahan NP
- Permasalahan NP-Complete
- Permasalahan NP-Hard

Bagian 1. Mesin Turing

Mesin Turing (1)

Mesin Turing:



Mesin Turing adalah model perhitungan matematika yang mendefinisikan mesin abstrak yang memanipulasi simbol pada pita sesuai dengan tabel aturan. Terlepas dari kesederhanaan modelnya, dengan algoritma komputer apa pun, Mesin Turing mampu mengimplementasikan logika algoritma tersebut. (*wikipedia*)

Mesin Turing (2)

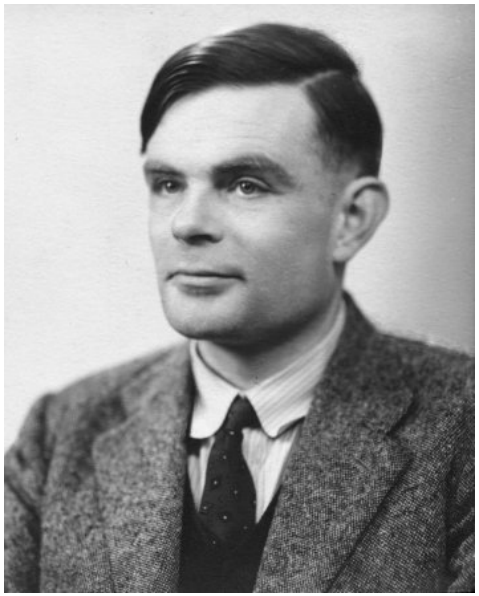


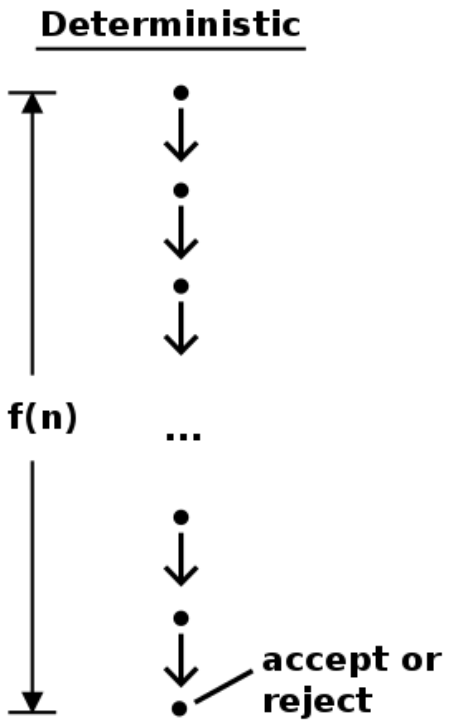
Figure: Alan Mathison Turing, (23 Juni 1912 7 Juni 1954), seorang matematikawan, ahli logika, kriptoanalis, dan ilmuwan komputer Inggris.

Bagian 2. Algoritma deterministik dan non-deterministik

Algoritma deterministik (1)

Definisi

Algoritma deterministik *adalah algoritma yang, dengan input tertentu, akan selalu menghasilkan* **output yang sama**, *dengan mesin yang mendasarinya selalu melewati urutan status yang sama.*



Algoritma deterministik (2)

Contoh: Sequential search.
Diberikan array n bilangan bulat $(a_1, a_2, \ldots, a_n)$. Kita ingin menemukan maksimum array.

Algorithm 1 Finding maximum of an array of integers

```
1: procedure MAX(A[1..n])
2:     max ← a_1
3:     for i = 2 to n do
4:         if a_i > max then
5:             max ← a_i
6:         end if
7:     end for
8: end procedure
```

Kompleksitas waktu: $\mathcal{O}(n)$

Algoritma non-deterministik (1)

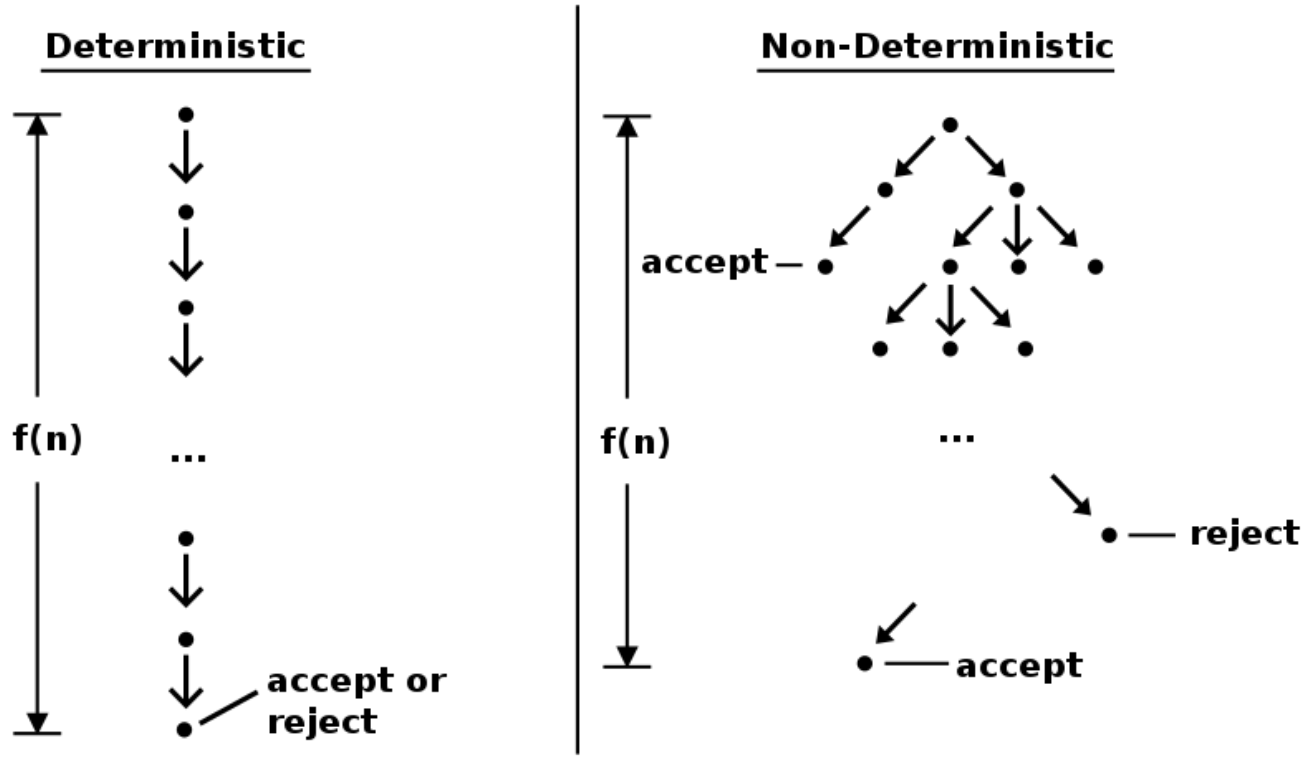
Definisi

Algoritma non-deterministik adalah prosedur dua tahap yang mengambil instance I dari masalah keputusan sebagai inputnya dan melakukan hal berikut.

- *Tahap non-deterministik ("guessing"): Sebarang string S dihasilkan yang dapat dianggap sebagai solusi kandidat untuk instance I yang diberikan.*
- *Tahap deterministik ("verifikasi"): Algoritma deterministik mengambil I dan S sebagai input dan outputnya jika S mewakili solusi untuk instance I. (Jika S bukan solusi untuk membuat instance I, algoritma akan mengembalikan no atau diizinkan untuk tidak berhenti sama sekali.)*

Sebuah algoritma non-deterministik memecahkan masalah keputusan jika dan hanya jika "*untuk setiap yes-instance dari masalah itu, algorima memberikan jawaban 'ya' pada beberapa eksekusi*". Dalam hal ini, kita memerlukan algoritma non-deterministik untuk dapat 'menebak ' solusi setidaknya sekali dan untuk dapat memverifikasi validitasnya. (Di samping itu, algoritma tidak pernah memberikan jawaban 'ya' pada *no-instance*).

Algoritma non-deterministik (2)

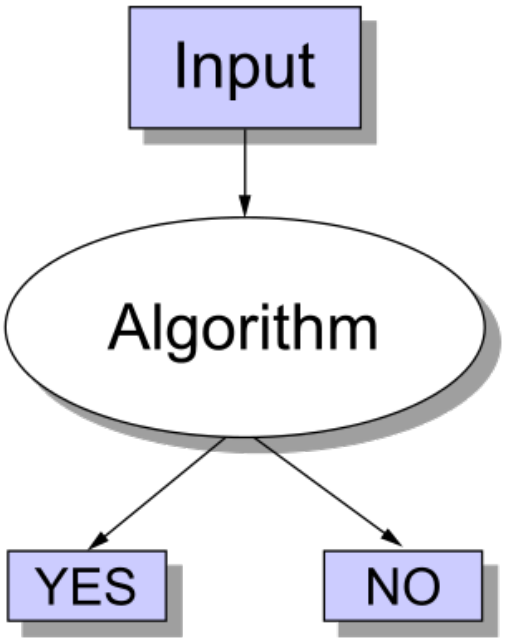


Bagian 3. Permasalahan keputusan

(*decidability & non-decidability*)

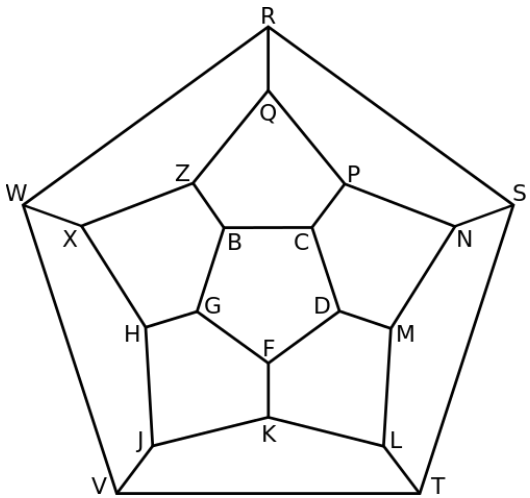
Masalah keputusan (*decision problems*)

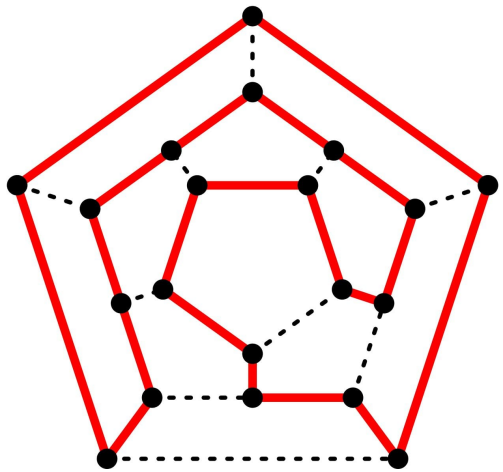
Masalah keputusan (*decision problems*) adalah masalah dengan jawaban ya/tidak.



Contoh masalah keputusan

1. Permasalahan sirkuit Hamilton: Tentukan apakah graf yang diberikan memiliki sirkuit Hamilton (yakni, lintasan yang dimulai dan berakhir pada simpul yang sama dan melewati semua simpul lainnya tepat satu kali).





2. Versi masalah **keputusan** TSP: Diberikan bilangan bulat positif ℓ, tujuanya adalah memutuskan apakah banyakya sirkuit Hamilton adalah paling banyak ℓ.

Permasalahan keputusan

Apakah *setiap* masalah keputusan dapat diselesaikan dalam waktu polinomial?

Definisi

Masalah keputusan yang dapat diselesaikan oleh suatu algoritma disebut masalah yang decidable. Masalah keputusan yang tidak dapat diselesaikan sama sekali oleh algoritma apa pun disebut masalah yang undecidable.

- **Decidable problem**: jika ada Mesin Turing yang berhenti pada setiap input dengan jawaban "yes" atau "no".
- **Undecidable problem**: jika kita tidak dapat membangun algoritma yang dapat menjawab masalah dengan benar dalam waktu yang terbatas, yaitu akan selalu ada kondisi yang akan menyebabkan Mesin Turing berputar tanpa batas tanpa memberikan jawaban sama sekali.

Bagian 3. Halting problem

Halting problem (1)

Contoh masalah yang dapat diputuskan:???

Contoh masalah yang tidak dapat diputuskan: Halting problem

Problem (Halting problem (Turing, 1936))

Diberikan sebuah program komputer dan inputnya, tentukan apakah program akan berhenti pada input tersebut atau terus mengerjakannya tanpa batas waktu.

Halting problem (2)

Halting problem adalah permasalahan yang *undecidable*.

Proof.

Untuk kontradiksi, asumsikan bahwa A adalah algoritma yang menyelesaikan halting problem. Artinya, untuk setiap program P dan masukan I:

$$A(P, I) = \begin{cases} 1, & \text{jika program } P \text{ berhenti } (\textit{halt}) \text{ pada input } I \\ 0, & \text{jika program } P \text{ tidak berhenti pada input } I \end{cases}$$

Pertimbangkan program P sebagai input untuk dirinya sendiri dan gunakan output dari algoritma A untuk pasangan (P, P) untuk membuat program Q sebagai berikut:

$$Q(P) = \begin{cases} \text{halts}, & \text{if } A(P, P) = 0, \text{ i.e., program } P \text{ tidak berhenti pada input } P \\ \text{does not halt}, & \text{program } P \text{ berhenti pada input } P \end{cases}$$

Substitutsikan Q untuk P menghasilkan:

$$Q(Q) = \begin{cases} \text{halts}, & \text{if } A(Q, Q) = 0, \text{ i.e., program } Q \text{ tidak berhenti pada input } Q \\ \text{does not halt}, & \text{if } A(Q, Q) = 1, \text{ i.e., program } Q \text{ berhenti pada input } Q \end{cases}$$

Oleh karena itu, ini merupakan kontradiksi karena tidak satu pun dari dua hasil untuk program Q yang mungkin.

Bagian 4. Tractability

Masalah di P

Definisi (Permasalahan polinomial)

Kelas P adalah kelas masalah keputusan yang dapat diselesaikan dalam waktu polinomial dengan algoritma deterministik. Kelas permasalahan ini disebut kelas polinomial.

Contoh permasalahan polinomial

- *Searching* $\rightarrow T(n) = \mathcal{O}(n), T(n) = \mathcal{O}(\log n)$
- *Sorting* $\rightarrow T(n) = \mathcal{O}(n^2), T(n) = \mathcal{O}(n \log n)$
- Perkalian matriks $\rightarrow T(n^3) = \mathcal{O}(n), T(n) = \mathcal{O}(n^{2.83})$

Contoh masalah non-polinomial

- TSP $\rightarrow T(n) = \mathcal{O}(n!)$
- Integer knapsack problem $\rightarrow T(n) = \mathcal{O}(2^n)$
- Masalah pewarnaan graf, dll.

Tractable & intractable problems

Definisi

Kita mengatakan bahwa suatu algoritma memecahkan masalah dalam waktu polinomial jika efisiensi waktu kasus terburuknya termasuk $\mathcal{O}(p(n))$ di mana $p(n)$ adalah polinomial dari ukuran input masalah n.

(*Perhatikan bahwa karena kita menggunakan notasi $\mathcal{O}$ di sini, karena masalah yang dapat diselesaikan dalam waktu logaritmik juga dapat diselesaikan dalam waktu polinomial.*)

Masalah yang dapat diselesaikan dalam waktu polinomial disebut tractable, dan masalah yang tidak dapat diselesaikan dalam waktu polinomial disebut intractable.

- Waktu polinomial: $\mathcal{O}(n^k)$, $\mathcal{O}(1)$, $\mathcal{O}(n \log n)$
- Tidak dalam waktu polinomial: $\mathcal{O}(2^n)$, $\mathcal{O}(n!)$, $\mathcal{O}(n^n)$

Permasalahan yang *decidable* namun *intractable*

- Masalah sirkuit Hamilton: Tentukan apakah suatu graf tertentu memiliki sirkuit Hamilton.
- Traveling salesman problem: Temukan tur terpendek melalui n kota dengan jarak bilangan bulat positif yang diketahui di antaranya.
- Knapsack problem: Temukan subset dengan nilai optimum dari n objek dengan bobot dan nilai bilangan bulat positif yang diberikan yang sesuai dengan ransel kapasitas bilangan bulat positif yang diberikan.
- Partition problem: Diberikan n bilangan bulat positif, tentukan apakah mungkin untuk mempartisinya menjadi dua himpunan bagian terpisah dengan jumlah yang sama.
- Graph-coloring problem: Untuk graf tertentu, carilah *bilangan kromatik*, yang merupakan jumlah warna terkecil yang perlu diberikan pada simpul-simpul graf sehingga tidak ada dua simpul bertetangga yang memiliki warna yang sama.
- Integer linear programming problem: Temukan nilai maksimum (atau minimum) dari fungsi linier dari beberapa variabel bernilai bilangan bulat yang tunduk pada batasan terbatas dalam bentuk persamaan dan pertidaksamaan linier.

Bagian 5. Permasalahan NP dan NP-Complete

Permasalahan NP

NP: non-deterministic polynomial (**bukannya** "non-polynomial time algorithm")

Definisi (Algoritma polinomial non-deterministik)

Algoritma non-deterministik waktu-polinomial merupakan algoritma non-deterministik yang tahapan verifikasinya dapat dilakukan dalam waktu polinomial.

Verifikasi pada waktu polinomial berarti:

- Jika diberikan kandidat solusi, kita dapat memeriksa apakah jawabannya benar/salah dalam waktu polinomial.
- Perhatikan bahwa ini setara dengan "menemukan solusi dalam waktu yang lama".

Contoh: Dalam versi keputusan TSP, diberikan solusi TSP dari graf, bilangan bulat positif k, kita dapat memeriksa dalam waktu polinomial jika solusinya memang benar merupakan TSP dan memiliki bobot $\leq k$.

Permasalahan NP

Definisi (Kelas *NP*)

Kelas NP adalah kelas masalah keputusan yang dapat diselesaikan dengan algoritma polinomial non-deterministik. Kelas masalah ini disebut kelas polinomial non-deterministik.

Catatan.

- Sebagian besar masalah keputusan ada di *NP*, dan $P \subseteq NP$
  - sebab jika masalah ada di P, kita dapat menggunakan algoritma deterministik polinomial yang menyelesaikannya dalam tahap verifikasi dari algoritma non-deterministik yang hanya mengabaikan string S yang dihasilkan dalam fase non-deterministiknya (yakni fase "*guessing*' ').
- $NP \not\subseteq P$, karena beberapa masalah ada di NP tetapi tidak di P.
  - Contoh: Masalah sirkuit Hamilton, versi keputusan dari TSP, knapsack, dan masalah pewarnaan graf, dll.
- Terdapat beberapa masalah yang tidak ada di *NP*. Contoh: *Halting problem*.

Apakah $P = NP$?

Ini merupakan pertanyaan terbuka paling penting pada bidang Ilmu Komputer Teoretis:

$$P \stackrel{?}{=} NP$$

- $P = NP$ akan menyiratkan bahwa masing-masing dari ratusan masalah keputusan kombinatorial yang sulit dapat diselesaikan dengan algoritma waktu polinomial (*ini masih terbuka meskipun banyak ilmuwan komputer telah berupaya selama bertahun-tahun*).
- Banyak masalah keputusan terkenal dikenal sebagai "NP-Complete" $\rightarrow$ lebih meragukan kemungkinan bahwa $P = NP$.

Millennium Prize Problems

Tujuh soal matematika terkenal yang dipilih oleh Clay Mathematics Institute pada tahun 2000. Clay Mathematics Institute menjanjikan hadiah US$ 1 juta untuk solusi yang tepat dari setiap soal.

1. Birch and Swinnerton-Dyer conjecture
2. Hodge conjecture
3. Navier-Stokes existence and smoothness
4. P versus NP problem
5. Poincaré conjecture (solved)
6. Riemann hypothesis
7. Yang-Mills existence and mass gap

Bagian 6. Reduksi ke masalah NP-Complete

Definisi permasalahan NP-Complete

Secara informal, sebuah masalah *NP*-complete adalah masalah di *NP* yang sama sulitnya dengan masalah lain di kelas ini. Sesuai definisi, masalah lain di *NP* *dapat direduksi* menjadi masalah tersebut dalam waktu polinomial.

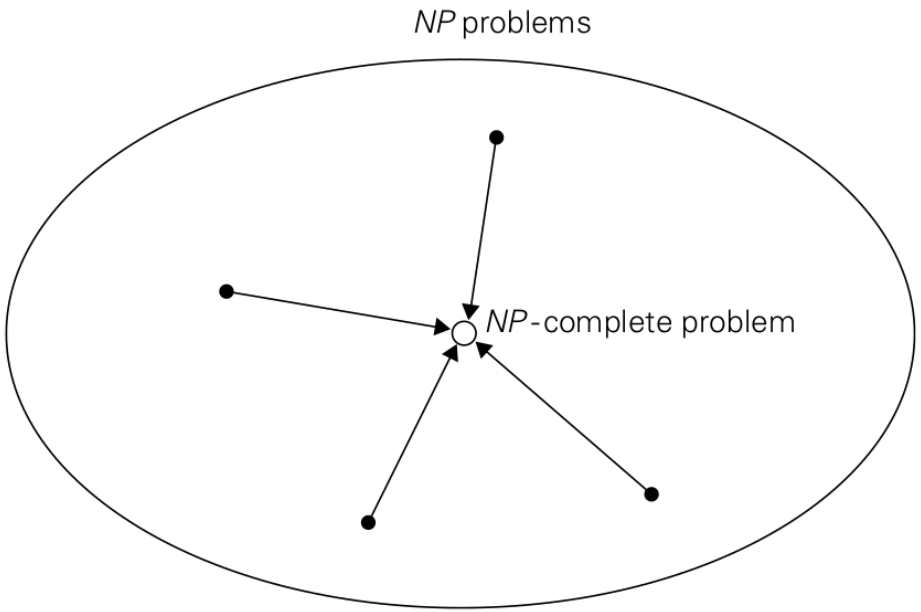


Figure: Gagasan tentang masalah *NP*-Complete. Reduksi dalam waktu polinomial dari masalah *NP* menjadi masalah NP-complete ditunjukkan oleh tanda panah.

Definisi reduksi polinomial

Definisi (Masalah reduksi polinomial)

Masalah keputusan D_1 dikatakan dapat direduksi secara polinomial menjadi masalah keputusan D_2, jika terdapat fungsi t yang mengubah instance dari D_1 menjadi instance dari D_2 sehingga:

- *t memetakan semua yes-instance dari D_1 ke yes-instance dari D_2, dan semua no-instance dari D_1 ke no-instance dari D_2*
- *t dapat dihitung dengan algoritma dalam waktu polinomial, yaitu $t \in NP$.*

Implikasi: Jika sebuah permasalahan D_1 dapat direduksi secara polinomial menjadi beberapa masalah D_2 yang dapat diselesaikan dalam waktu polimial, maka permasalahan D_1 juga dapat diselesaikan dalam waktu polinomial.

$$D_1 \xrightarrow[\text{in } P]{\text{reduced to}} D_2 \quad \text{in } P$$

Reduksi polinomial pada permasalahan NP-Complete

Definisi (*NP*-Complete problem)

Masalah keputusan D dikatakan NP-complete jika:

- *termasuk dalam kelas NP*
- *setiap masalah dalam NP dapat direduksi secara polinomial menjadi D*

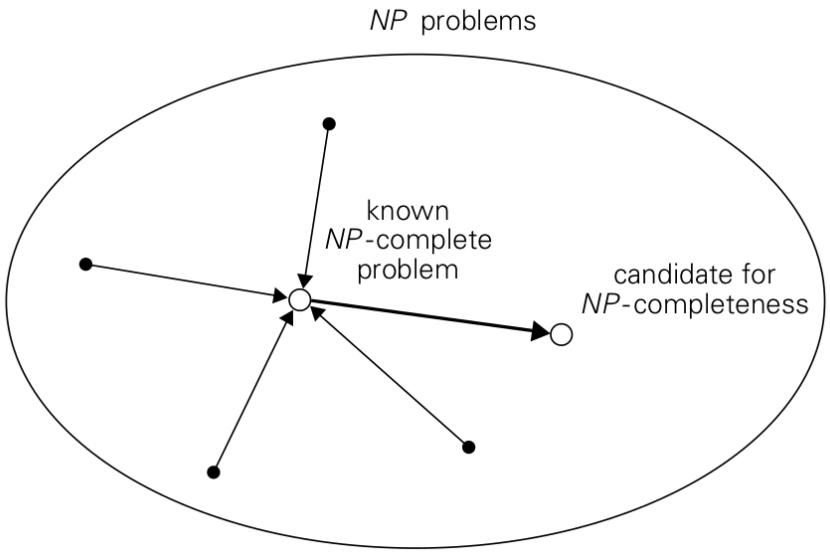


Figure: Membuktikan *NP*-complete dengan reduksi

- Jika X adalah NPC dan X dapat diselesaikan dalam waktu polinomial, maka semua masalah *NP* dapat diselesaikan dalam waktu polinomial;
- yaitu jika X dapat diselesaikan dalam waktu polinomial, maka $P = NP$.

Daftar permasalahan NP-Complete (4)

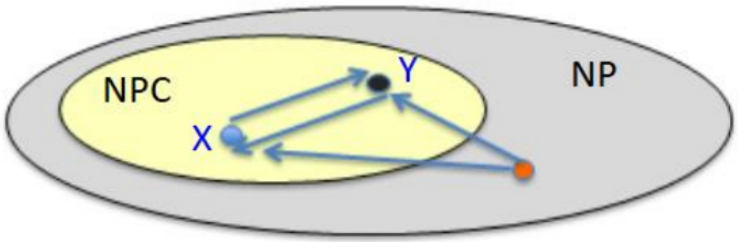
Daftar permasalahan NP-Complete

- Boolean satisfiability problem (SAT)
- *Decision*-version TSP
- Masalah sirkuit Hamilton
- Partition problem
- Clique problem
- Decision-version dari masalah pewarnaan graf
- Vertex cover problem
- Decision-version dari Knapsack problem

Reduksi polinomial (1)

Sifat Permasalahan NP-Complete

- Masalah X adalah *NPC* jika ada masalah di *NP* yang dapat dikurangi (diubah) menjadi X dalam waktu polinomial.
- Dua masalah X dan Y dalam *NPC* dapat direduksi satu sama lain dalam waktu polinomial.
  - X dapat direduksi menjadi Y dalam waktu polinomial
  - Y dapat direduksi menjadi X dalam waktu polinomial



Reduksi polinomial (2)

Bagaimana menunjukkan bahwa masalah X adalah *NPC*?

- Tunjukkan bahwa X adalah *NP*
- Pilih masalah Y dari kumpulan masalah *NPC*
- Bangun algoritma reduksi yang mereduksi turunan masalah Y menjadi turunan masalah Z.

Reduksi polinomial (3)

Contoh: Masalah sirkuit Hamilton dapat direduksi secara polinomial menjadi versi keputusan TSP

- **Masalah sirkuit Hamilton**: Tentukan apakah graf yang diberikan memiliki sirkuit Hamilton – lintasan yang berawal dan berakhir pada simpul yang sama dan melewati semua simpul lainnya tepat satu kali.

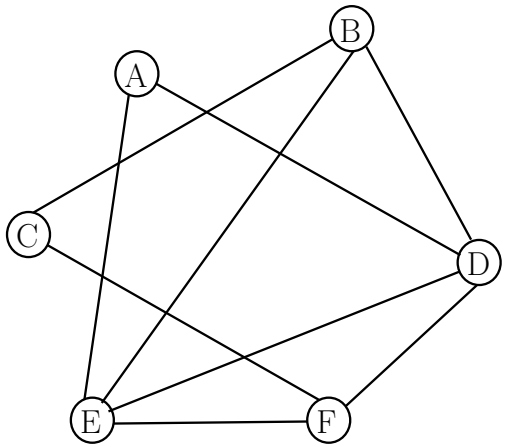
- **TSP-decision problem**: Diberikan grafik dan jarak antara pasangan simpul, dan bilangan bulat positif ℓ, tugasnya adalah memutuskan apakah grafik memiliki tur paling banyak ℓ.

Masalah sirkuit Hamilton $\xrightarrow{\text{polynomially reducible}}$ TSP-decision

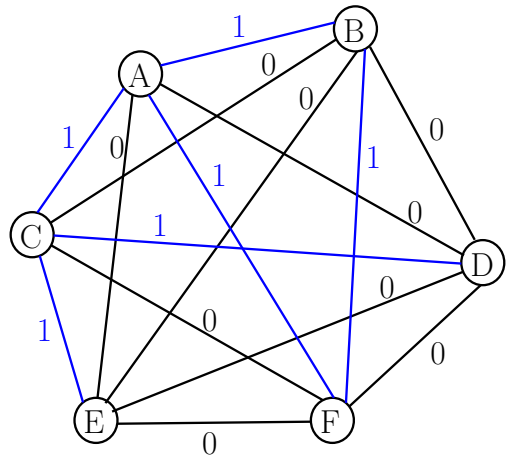
Kita memetakan graf G dari contoh tertentu dari masalah sirkuit Hamilton ke graf lengkap berbobot G yang mewakili turunan dari TSP.

Reduksi:

- Tetapkan 0 sebagai bobot untuk setiap sisi dalam G dan tambahkan sisi dengan bobot 1 di antara setiap pasangan simpul yang tidak berdekatan dalam G.



G



G'

med

Masalah sirkuit Hamilton $\xrightarrow{\text{polynomially reducible}}$ TSP-decision

- **G memiliki sirkuit Hamilton jika ada sirkuit dalam G' melewati semua simpul tepat satu kali, dan memiliki panjang ≤ 0** (yaitu memiliki solusi untuk turunan TSP di mana $k = 0$).

  1. **Jika ada sirkuit yang melewati semua simpul tepat satu kali, dan memiliki panjang ≤ 0 di G'**, sirkuit hanya berisi sisi yang awalnya ada di G. (Tepi baru di G' memiliki bobot 1 dan karenanya tidak dapat menjadi bagian dari sirkuit dengan panjang ≤ 0.)

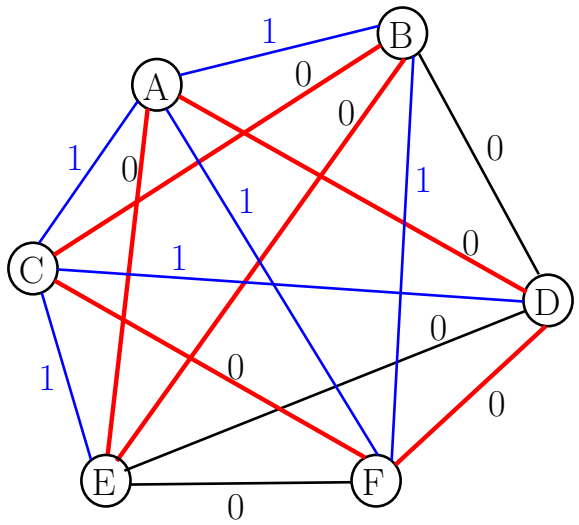
     $\Rightarrow$ Terdapat sirkuit Hamilton di G.

  2. **Jika ada sirkuit Hamilton di G**, itu membentuk sirkuit di G' dengan panjang $= 0$, karena bobot semua sisi adalah 0.

     $\Rightarrow$ Ada solusi untuk TSP di G' dengan panjang ≤ 0.

Masalah sirkuit Hamilton $\xrightarrow{\text{polynomially reducible}}$ TSP-decision

Contoh:

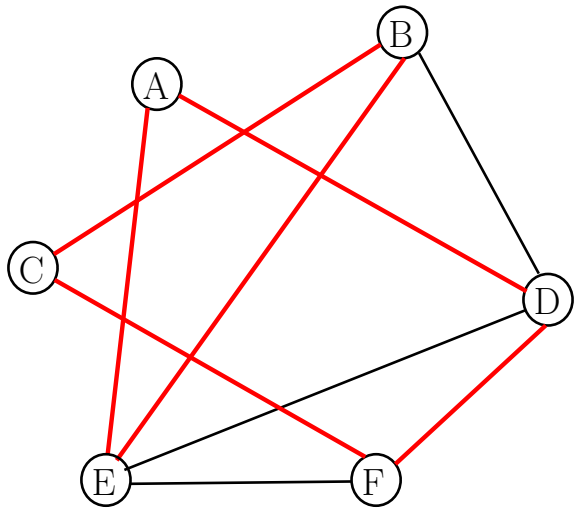


G'

Figure: G' memiliki sirkuit yang melewati semua simpul tepat satu kali dengan panjang ≤ 0.

Masalah sirkuit Hamilton $\xrightarrow{\text{polynomially reducible}}$ TSP-decision

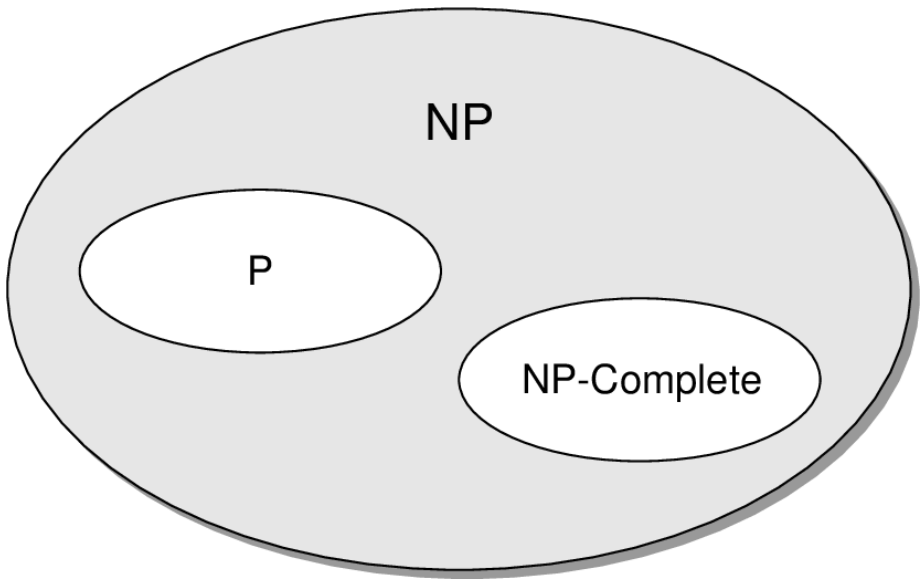
Contoh:



G

Figure: G' memiliki sirkuit melewati semua simpul. Ini adalah sirkuit Hamilton di G

Diagram *P*, *NP*, dan *NP*-Complete



Bagian 7. CNF-satisfiability problem

Permasalahan *CNF-satisfiability*

- x_1, x_2, x_3, dan x_4 adalah Variabel Boolean yang akan ditetapkan (nilai 0 atau 1)
- $\neg$ berarti negation (logika *not*)
- $\wedge$ berarti conjunction (logika *and*)
- $\vee$ berarti disjunction (logika *or*)
- Literal adalah variabel atau negasinya, contoh: x_i and $\neg x_i$
- Clause adalah disjungsi ($\wedge$) dari literal, misalnya: $x_i \vee x_j$
- Conjunctive Normal Form (CNF) adalah konjungsi dari klausa

  Contoh: $(x_1 \vee x_2) \wedge (\neg x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_4)$

Permasalahan *CNF-satisfiability*

Definisi

Satisfiability Problem (SAT) adalah masalah kombinatorial klasik. Diberikan rumus Boolean dari n variabel:

$$f(x_1, x_2, \ldots, x_n)$$

Masalahnya adalah **bagaimana menemukan nilai variabel seperti itu, dimana formula mengambil nilai True**.

CNF Satisfiability Problem (CNF-SAT) adalah varian dari Satisfiability Problem, dimana rumus Boolean di atas ditentukan dalam Conjunctive Normal Form (CNF).

Permasalahan *CNF-satisfiability*

Input: Ekspresi atas variabel Boolean dalam *conjunctive normal form (CNF)*.

Pertanyaan: Apakah ekspresinya *satisfiable* yaitu, dapatkah kita memberi setiap variabel nilai (benar atau salah) sehingga ekspresi menjadi benar?

Permasalahan *CNF-satisfiability*

Input: Ekspresi atas variabel Boolean dalam *conjunctive normal form (CNF)*.

Pertanyaan: Apakah ekspresinya *satisfiable* yaitu, dapatkah kita memberi setiap variabel nilai (benar atau salah) sehingga ekspresi menjadi benar?

Contoh: $(x_1 \vee x_2) \wedge (\neg x_2 \vee x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_4)$

Formula ini dikatakan satisfiable karena pada $x_1 = \mathit{True}$, $x_2 = \mathit{False}$, $x_3 = \mathit{False}$, dan $x_4 = \mathit{True}$, dibutuhkan nilai True.

Periksalah!

Permasalahan *CNF-satisfiability*

Teorema (Cook-Levin Theorem)

CNF-Satisfiability adalah permasalahan NP-complete.

Proof. Bukti dapat dibaca di
`https://en.wikipedia.org/wiki/CookLevin_theorem`

- Yang paling terkenal adalah buktinya Cook, menggunakan Mesin Turing karakterisasi *NP*.
- Ini merancang Mesin Turing yang memverifikasi ya-contoh SAT

Bagian 8. Permasalahan NP-Hard

NP-Hard

Definisi (Permasalahan NP-Hard)

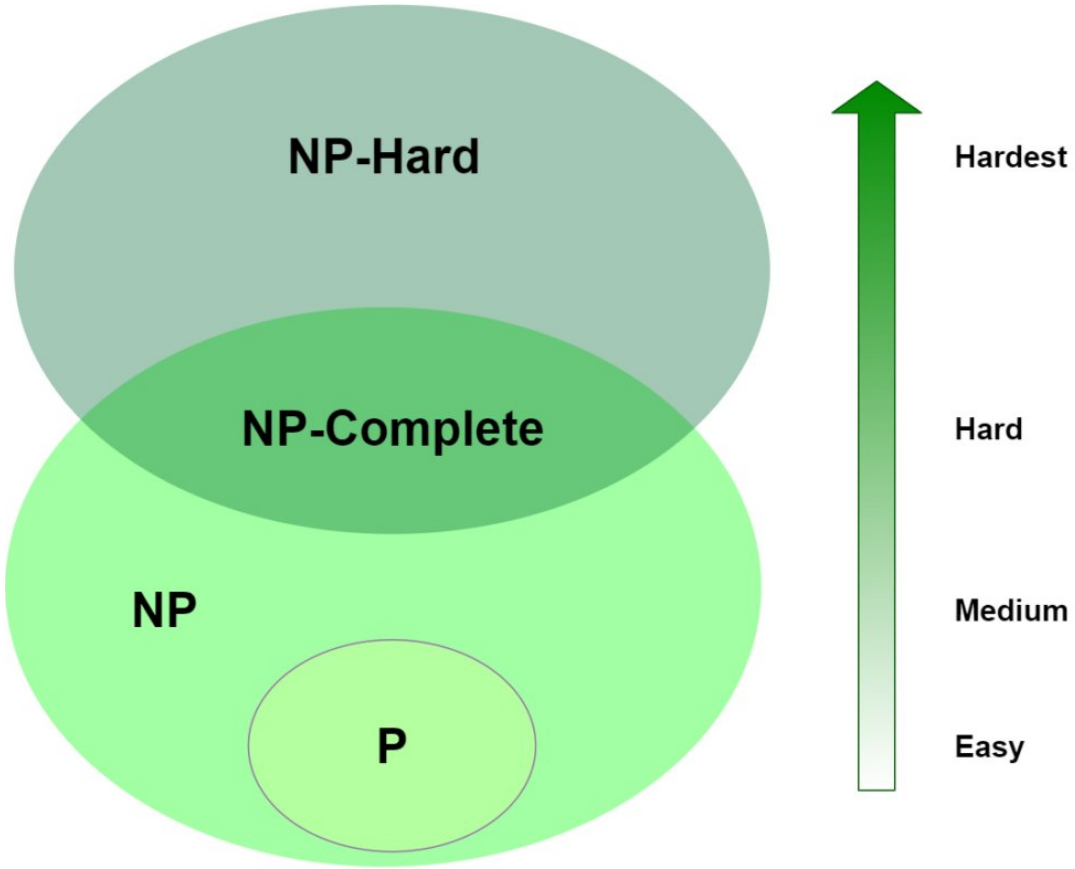
Masalah keputusan H dikatakan NP-hard jika untuk setiap masalah L di NP, ada pengurangan banyak-satu waktu polinomial dari L ke H.

- Suatu masalah adalah NP-hard jika suatu algoritma untuk menyelesaikannya dapat diterjemahkan menjadi satu untuk memecahkan masalah NP apa pun.
- NP-hard berarti "setidaknya memiliki kesulitan setara dengan masalah NP apa pun" meskipun sebenarnya mungkin lebih sulit.
- Masalah NP-hard seringkali memiliki kompleksitas waktu eksponensial.

Contoh: TSP versi *non-decision*

Remark. Jika $P \neq NP$, maka masalah *NP*-Hard tidak dapat diselesaikan dalam waktu polinomial.

Diagram kelas kompleksitas



end of slide...