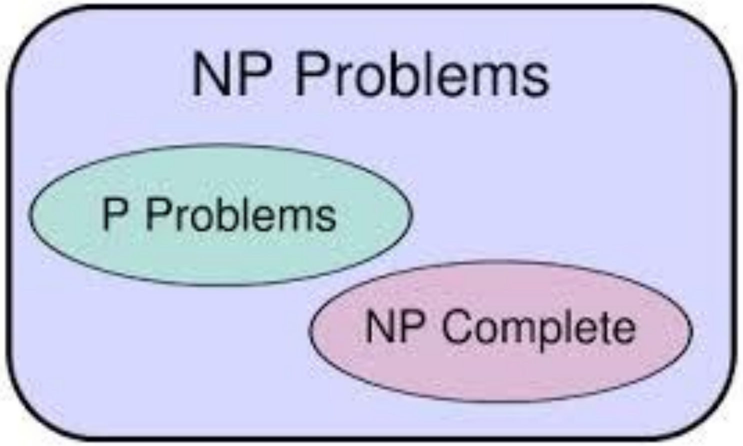


Teori P, NP, dan NP-Complete

(Bagian 1)

Bahan Kuliah IF2211 Strategi Algoritma

Oleh: Rinaldi Munir

Program Studi Teknik Informatika ITB

Ikhtisar

- Polynomial-time algorithm vs nonpolynomial-time algorithm
- Tractable problem vs intractable problem
- Solvable problem vs unsolvable problem
- Halting problem
- Deterministic vs nondeterministic algorithm
- Decision problem

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 kompleksitas waktunya dibatasi oleh fungsi polinom sebagai fungsi dari 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 kompleksitas waktunya dibatasi oleh fungsi non-polinom sebagai fungsi dari 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*).

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 bertambahnya ukuran masukan persoalan.

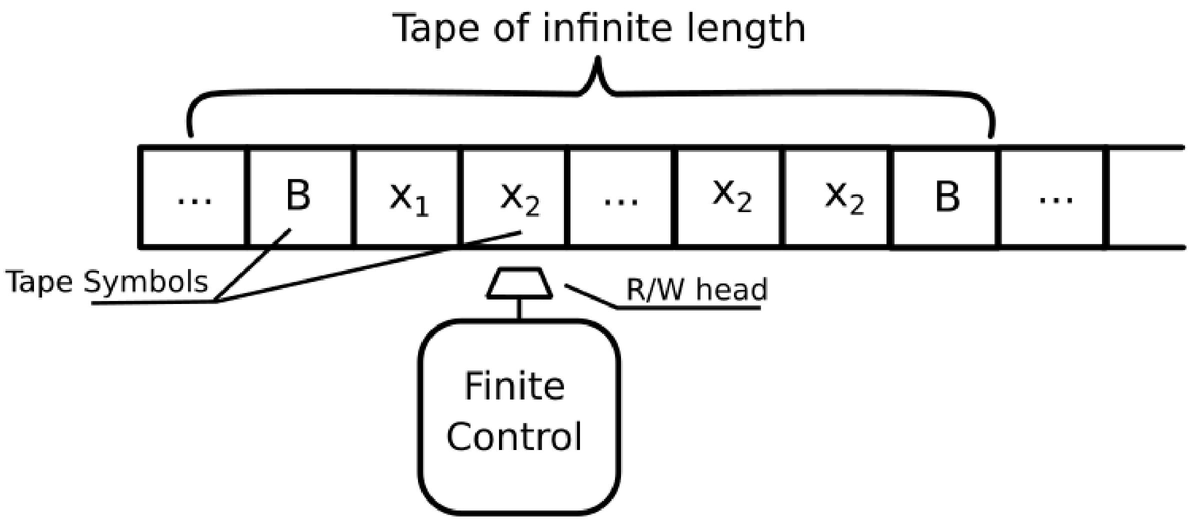
- Apa yang dimaksud dengan waktu yang wajar? Standar waktunya adalah *polynomial time*.
  - *Polynomial time*: $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$
  - *Not in polynomial time*: $O(2^n)$, $O(n^n)$, $O(n!)$ untuk n yang kecil

Solvable vs *Unsolvable Problem*

Dikaitkan dengan Mesin Turing, sebuah persoalan dikatakan:

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

- *Solvable problem* dibagi menjadi dua kategori:
  1. *Tractable*
  2. *Intractable*



- Adakah persoalan yang *unsolvable*? Ada. Contoh persoalan *unsolvable* 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

- Kode program berikut

```
i = 0
while (true) { i = i + 1}
```

tidak pernah berhenti (*infinite loop*)

- Sedangkan program

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

berhenti dengan cepat.

- Misalkan *A* adalah algoritma untuk menyelesaikan *halting problem*.
- *A* menerima input: (i) kode program *P* dan (ii) input untuk program *P*, yaitu *I*,

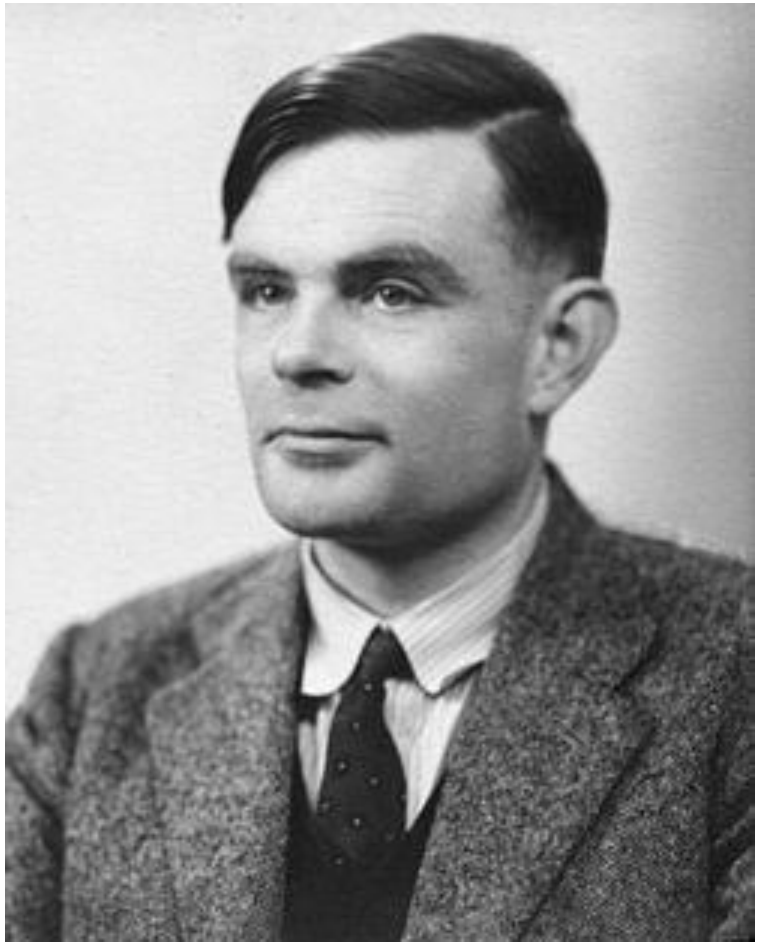
$A(P, I)$ = 1, jika program *P* berhenti untuk masukan *I*

= 0, jika program *P* tidak berhenti

- Turing membuktikan tidak ada algoritma *A* yang dapat memutuskan apakah program *P* berhenti ketika dijalankan dengan masukan *I* itu.

  → *Halting problem* tidak bisa diselesaikan → *unsolvable problem*

- **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

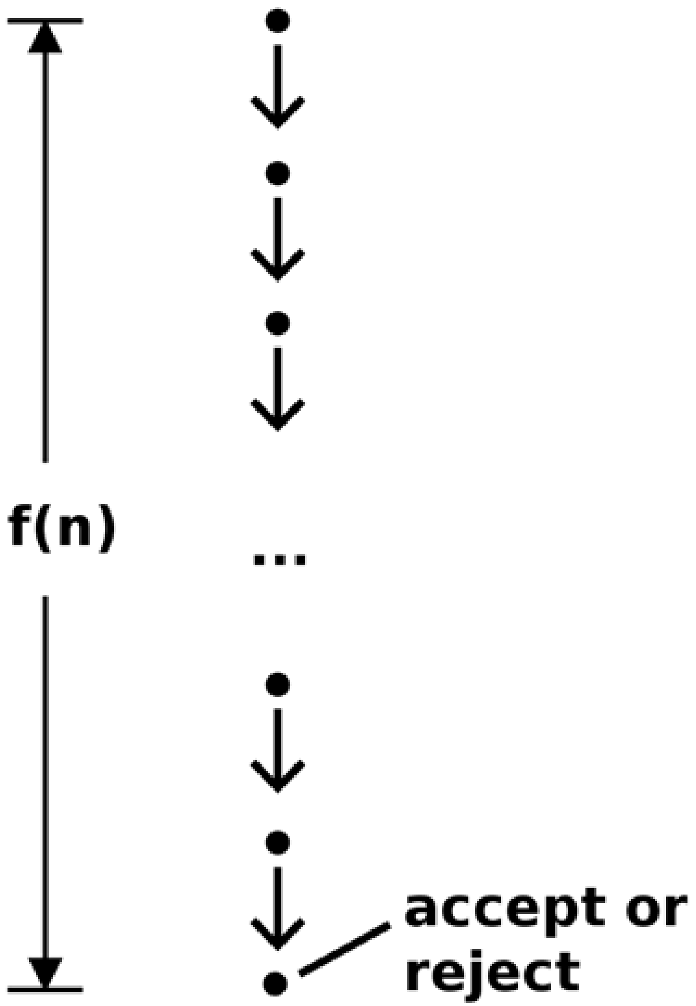
- Algoritma deterministik adalah algoritma yang dapat ditentukan dengan pasti aksi apa yang akan dikerjakan selanjutnya oleh algoritma tersebut.

…
aksi *k*-1
aksi *k*
aksi *k*+ 1
…

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

- Semua algoritma yang sudah kita pelajari sejauh ini adalah algoritma deterministik

Deterministic



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:

(1) k $\leftarrow$ 1

(2) **while** (A[k] $\neq$ x) **and** (k < n) **do**

    k $\leftarrow$ k + 1

  end

(3) **if** A[k] = x **then**

    return k

  else

    return -1

  end

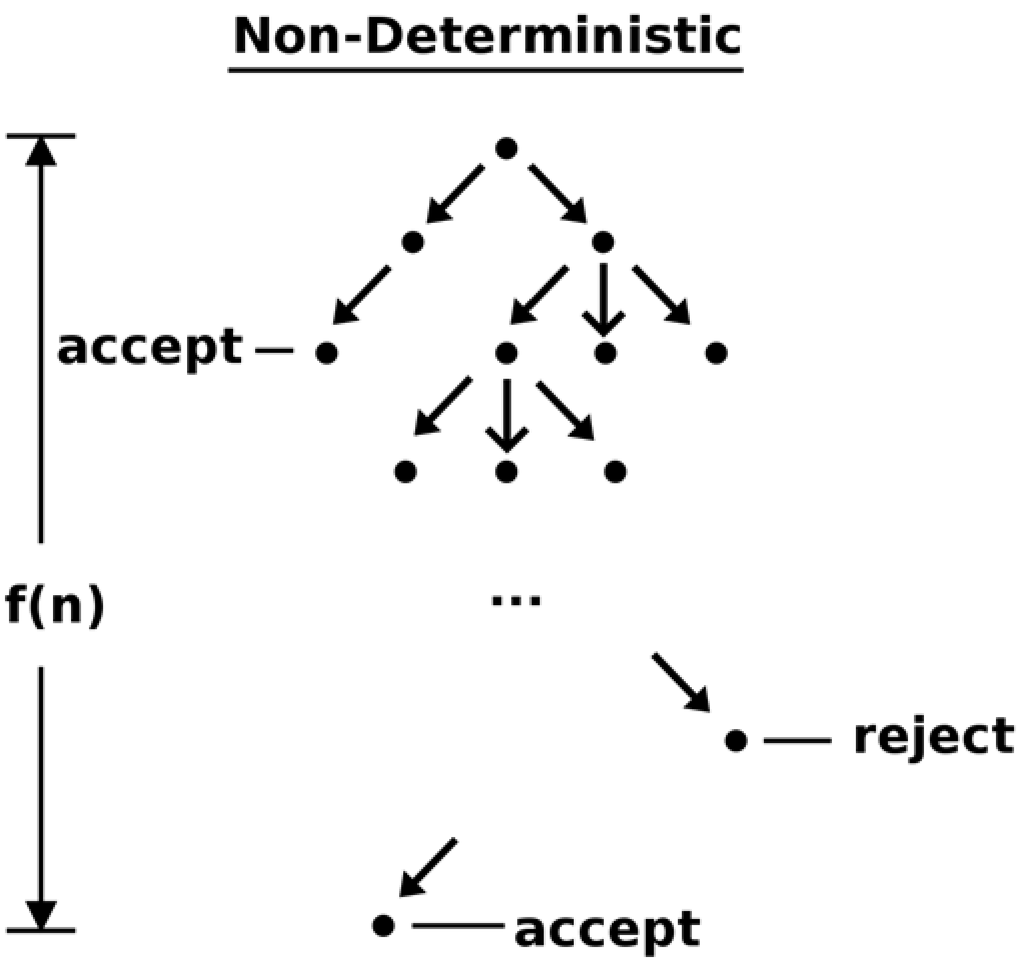
Kompleksitas waktu: O(n)

Algoritma Non-deterministik

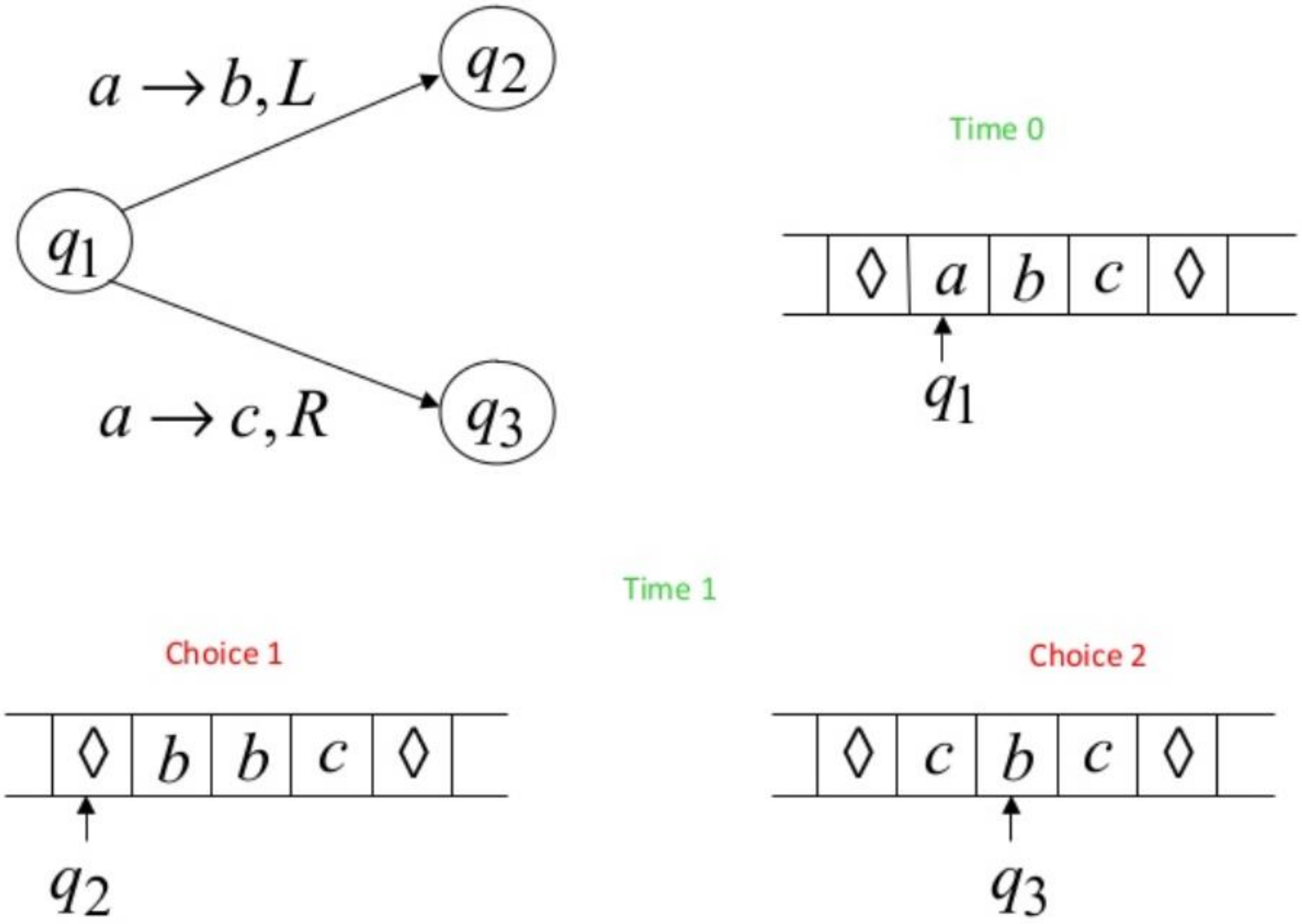
- Algoritma non-deterministik adalah algoritma yang di dalamnya berhadapan dengan beberapa pilihan aksi (opsi), dan algoritma memiliki kemampuan untuk menerka atau memilih sebuah aksi.

- Algoritma non-deterministik dijalankan mesin non-deterministik (komputer hipotetik, komputer bersifat imajiner atau teoritis).

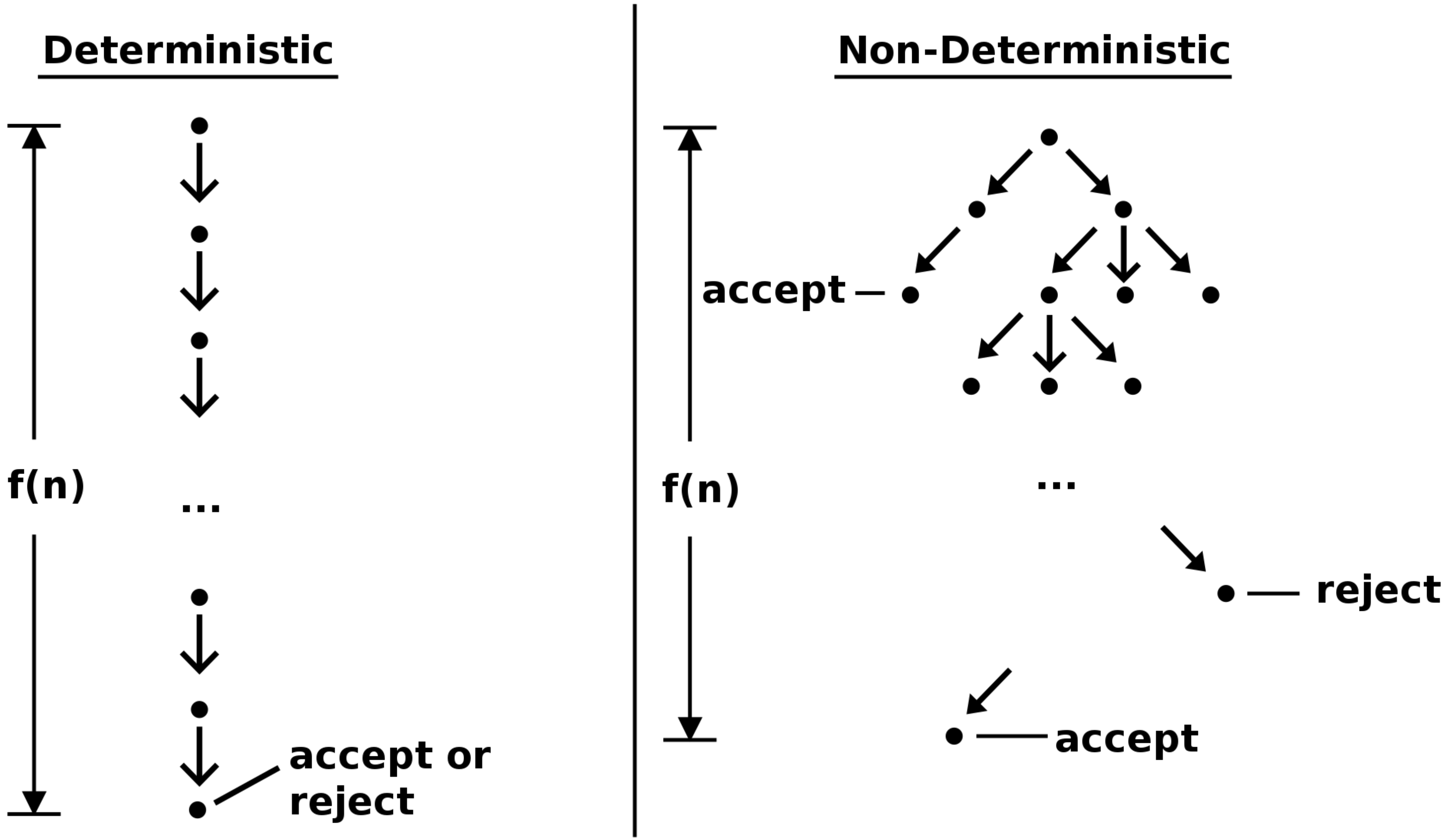
- Contoh: mesin turing nondeterministic.



Mesin Turing Nondeterminsitik



Perbedaan komputasi deterministik vs komputasi non deterministik



- Algoritma non deterministik dapat digunakan untuk menghampiri solusi persoalan-persoalan yang solusi eksaknya membutuhkan waktu komputasi yang mahal.

- Misalnya untuk menyelesaikan persoalan TSP, Knapsack, dll.

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)

else

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" (ekivalen dengan accept/reject, ada/tidak ada, bisa/tidak bisa)

  Contoh:

  1. Diberikan sebuah integer x.

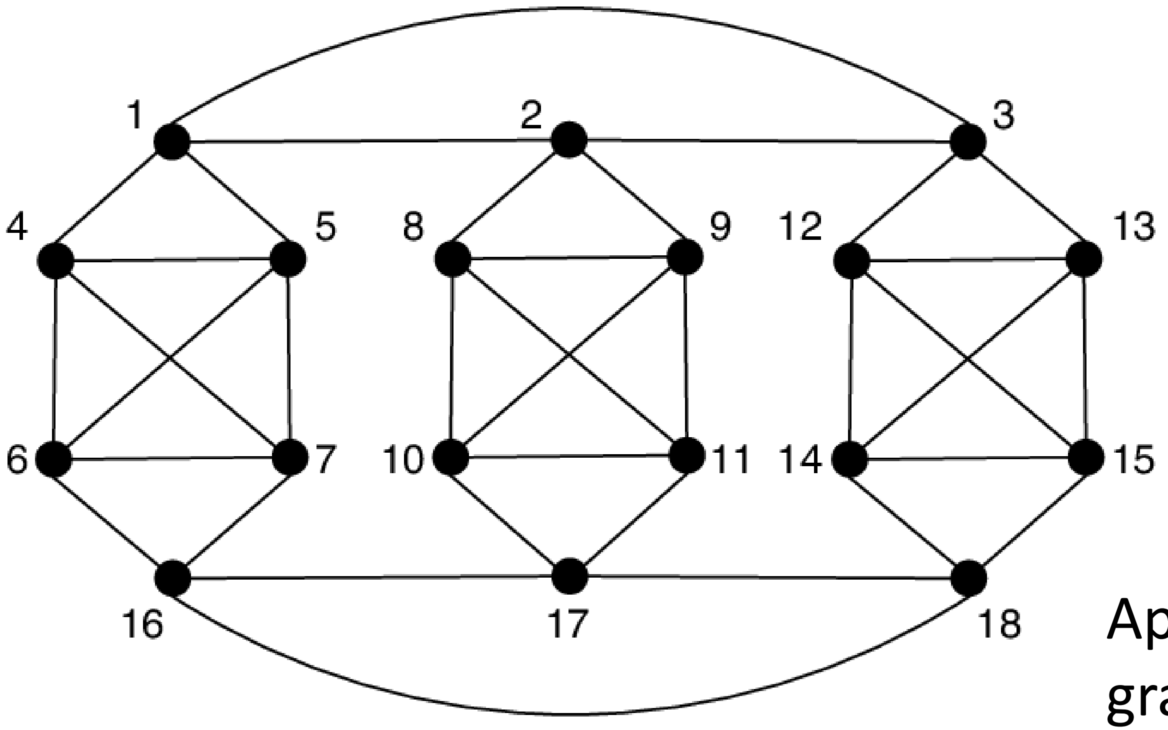
     Tentukan apakah elemen *x* terdapat di dalam tabel? Ada/tidak ada

  2. Diberikan sebuah integer x.

     Tentukan apakah x bilangan prima? Prima/tidak prima

Contoh-contoh persoalan keputusan lainnya

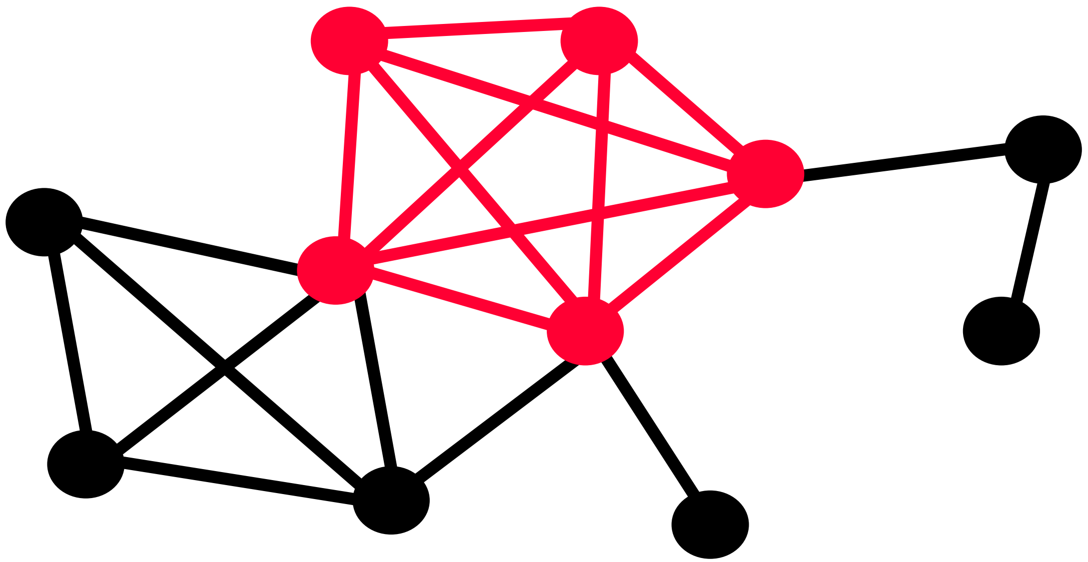
1. **Persoalan sirkuit Hamilton**



Apakah terdapat sirkuit Hamilton di dalam graf ini? (Yes/no)

2. Clique Problem

Sebuah *clique* adalah *subset* dari himpunan simpul di dalam graf yang semuanya terhubung.



Apakah terdapat clique yang jumlah simpulnya 5? (Yes/no)

Upagraf yang berwarna merah adalah sebuah *clique*

(Stephen Arthur Cook, 1971)

3. SAT (*Satisfiable Problem*)

- Diberikan X = {x_1, x_2, …, x_n} adalah himpunan peubah Boolean. Sebuah klausa (ekspresi) Boolean mengandung peubah atau negasi dari peubah. Sebuah koleksi klausa Boolean *C* dikatakan *satisfiable* jika terdapat *assignment* nilai-nilai kebenaran untuk X yang secara simultan membuat *C* bernilai *true*.

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

  x_i = {1, 0}, 1 = true, 0 = false

  jika *C* = 1 → *satisfied;* jika *C* = 0 → *not satisfied*

  misal x_1 = 1, x_2= 0, x_3=0 → *C* = (1' ∨ 0 ∨ 0') ∧ (0) ∧ (1 ∨ 0') = 0 → *not satisfied*

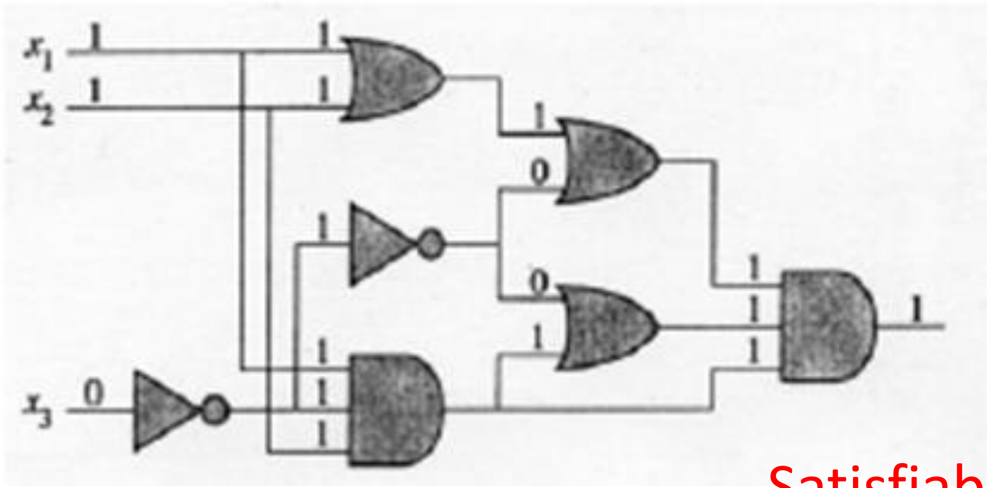
PERSOALAN SAT:
Diberikan *instance* persoalan, yaitu himpunan peubah Boolean *X* dan klausa *C*
Pertanyaan: apakah terdapat *assignment* nilai-nilai peubah sehingga *C satisfied*?
(Yes/No)

Example *Yes*

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

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

The answer is *Yes*



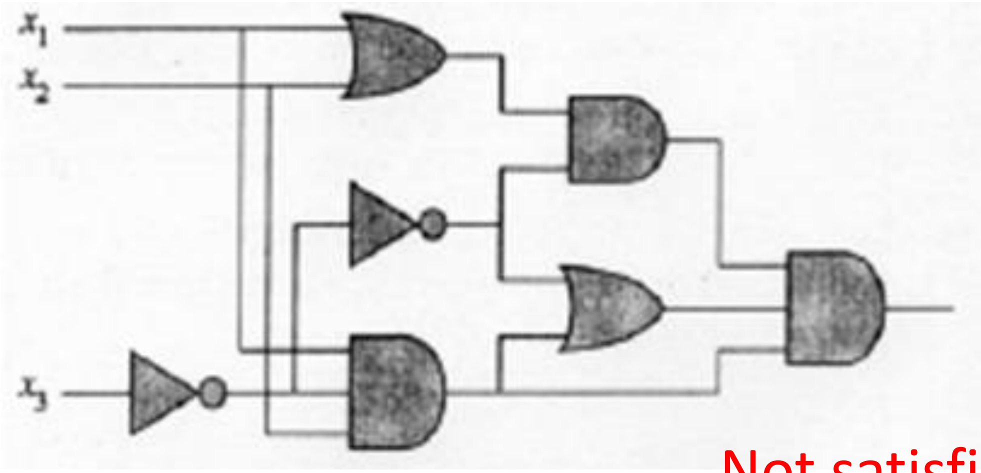
Satisfiable

Example *No*

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

There are no truth assignments that satisfies C'

The answer is *No*



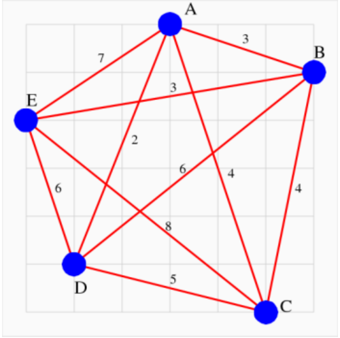
Not satisfiable

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

- Perhatikan beberapa persoalan berikut:

    1. Travelling Salesperson Problem (TSP)
    2. Knapsack Problem
    3. Graph Colouring

1. Travelling Salesperson Problem



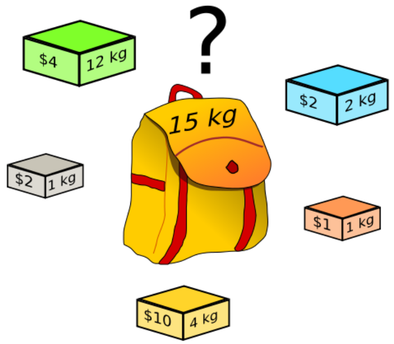
- Diberikan graf berarah dengan bobot (*weight*) pada setiap sisinya. Sebuah tur di dalam graf tersebut dimulai dari sebuah simpul,mengunjungi simpul lainnya tepat sekali dan kembali lagi ke simpul asalnya.

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

- *Travelling Salesperson Decision Problem* (TSDP): Apakah terdapat tur dengan total bobot sisinya ≤ *d*.

  → TSP decision problem.

2. *Knapsack Problem*



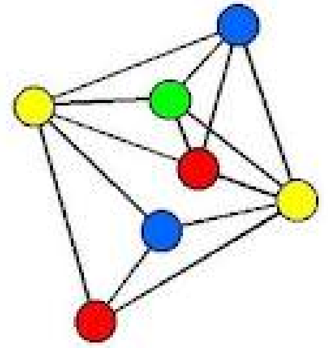
- Diberikan *n* buah objek dan sebuah *knapsack* dengan kapasitas *W*. Setiap objek memiliki bobot dan profit masing-masing.

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

- *Integer Knapsack Decision Problem*: Apakah dapat memasukkan objek-objek ke dalam *knapsack* namun tidak melebihi *W* tetapi total profitnya $\geq P$.

→ *Integer Knapsack decision problem*

3. Graph Colouring Problem



- *Graph-Colouring Optimization Problem*: Tentukan 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*: Apakah terdapat pewarnaan graf yang menggunakan paling banyak *m* warna sedemikian sehingga dua simpul bertetangga memiliki warna berbeda?

  → *Graph Colouring decision problem*

- 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 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 $d \leq 120$, atau “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 (misalnya) menghasilkan string *S*. String ini dapat dianggap sebagai sebuah terkaan solusi. String yang dihasilkan bisa saja tidak bermakna (*non-sense).*

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

Agoritma non-deterministik *Travelling Salesperson Decision Problem* (TSDP):

Persoalan: apakah terdapat tur di dalam graf dengan total bobot $\leq d$.

- Tahap menerka

  $S \leftarrow$ Terka(string)

- Tahap verifikasi

  S diverifikasi apakah merupakan sebuah tur lengkap, lalu periksa apakah total bobot semua sisinya lebih kecil atau sama dengan *d*

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

  return **true**

  else

  return **false**

  end

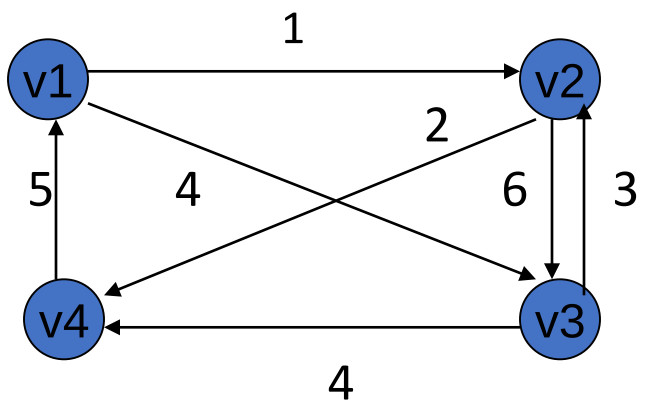
Kompleksitas waktu: O(n)

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

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

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

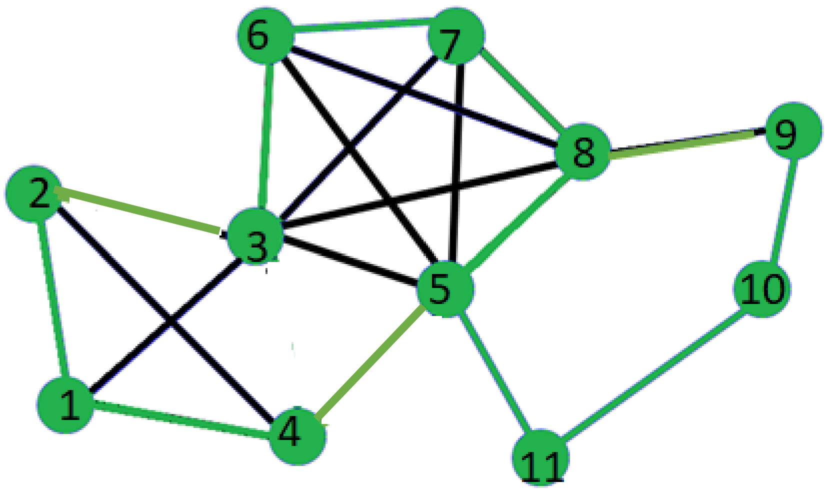
- Contoh untuk TSDP dengan d = 15:



S	Luaran	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 kali.



Algoritma non-deterministik:

1. Terkalah permutasi semua simpul
2. Verifikasi: Periksa apakah permutasi tersebut membentuk sirkuit. Jika benar, maka jawabannya adalah “true”, jika tidak maka jawabannya adalah “false”

BERSAMBUNG KE VIDEO BAGIAN 2