# 3.2 - Algoritma *Brute Force* (bagian 2)

[KOMS120403]

Desain dan Analisis Algoritma (2023/2024)

Dewi Sintiari

Prodi S1 Ilmu Komputer
Universitas Pendidikan Ganesha

Week 3 (March 2024)

# Daftar isi

- *Exhaustive search* (pencarian menyeluruh)
  1. The Traveling Salesman Problem
  2. Permasalahan 1/0 Knapsack
  3. Exhaustive search pada kriptografi
- Latihan: Masalah penugasan; Masalah partisi; *Magic square*
- Teknik heuristik

# Tujuan pembelajaran

Anda diharapkan mampu untuk:

1. Menjelaskan implementasi exhaustive search pada penyelesaian masalah algoritmik sederhana
2. Menjelaskan ide dasar dari teknik heuristik

# **Bagian 4:** *Exhaustive search* (pencarian menyeluruh)

# Exhaustive search

Exhaustive search secara sederhana merupakan sebuah pendekatan kasar untuk *masalah kombinatorial* (permutasi, kombinasi, himpunan bagian, dll.).

Catatan. Contoh soal kombinatorial adalah Traveling Salesman Problem, Knapsack problem, dll.; dan masalah non-kombinatorial adalah masalah Powering, Perkalian Matriks Persegi, dll.

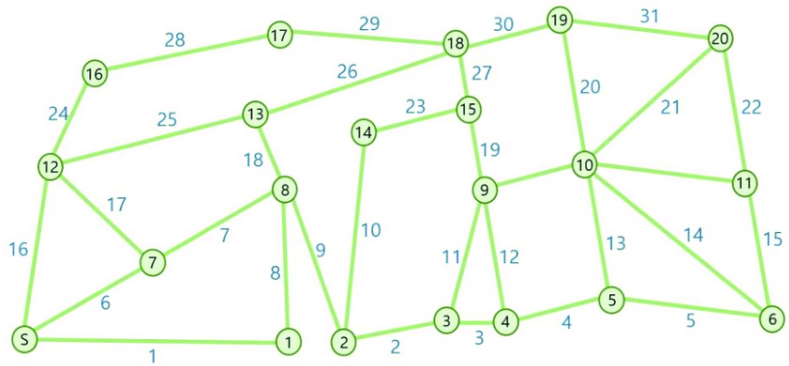
Catatan. Dalam banyak referensi, *exhaustive search* dianggap sama dengan brute force.

Baca buku Anany Levitin, lihat Bagian 3.4 (page 143)!

# 4.1. *Exhaustive search* untuk Permasalahan *Traveling Salesman Problem*

# 1. Traveling Salesman Problem (1) [page 142]

Permasalahan. Diberi $n$ kota dan jarak antara setiap pasangan kota, tentukan rute terpendek yang mengunjungi setiap kota tepat satu kali dan kembali ke kota asal?




Catatan. Kita dapat mengasumsikan bahwa graf input adalah **graf lengkap** (yakni setiap pasangan simpul digabungkan dengan sebuah sisi). Jika graf tidak lengkap (seperti pada gambar di atas, misalnya tidak ada sisi antara simpul 1 dan 7), maka kita dapat mengasumsikan bahwa sisi $(1, 7)$ ada, tetapi bobotnya adalah $\infty$ (sangat besar).

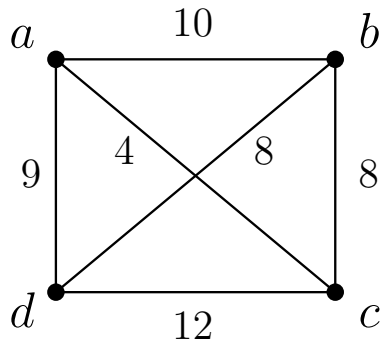
# 1. Traveling Salesman Problem (2)

Hamiltonian cycle adalah *cycle* (sirkuit) yang mengunjungi setiap simpul pada graf tepat satu kali. Masalah TSP setara dengan *menemukan sirkuit Hamilton dengan bobot minimum*.

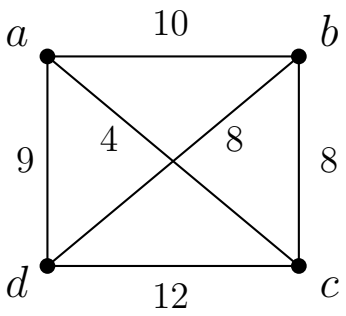
**Algoritma exhaustive search untuk TSP**

1. Menghitung semua sirkuit Hamilton dari graf lengkap $n$-simpul.
2. Evaluasi bobot setiap sirkuit Hamilton yang ditemukan pada langkah 1.
3. Pilih sirkuit Hamilton dengan bobot minimum.

Latihan. Terapkan algoritma di atas pada graf berikut!

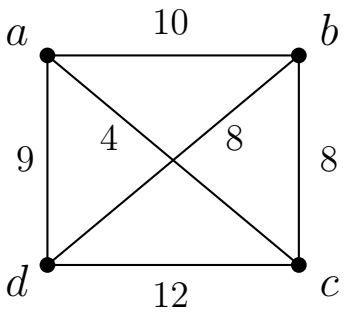


# 1. Traveling Salesman Problem (3)




| No. | Traveling route | Weight |
|---|---|---|
| 1. | $a \to b \to c \to d \to a$ | $10 + 8 + 12 + 9 = 39$ |
| 2. | $a \to b \to d \to c \to a$ | $10 + 8 + 12 + 4 = 34$ |
| 3. | $a \to c \to b \to d \to a$ | $4 + 8 + 8 + 9 = 29$ |
| 4. | $a \to c \to d \to b \to a$ | $4 + 12 + 8 + 10 = 34$ |
| 5. | $a \to d \to b \to c \to a$ | $9 + 8 + 8 + 4 = 29$ |
| 6. | $a \to d \to c \to b \to a$ | $9 + 12 + 8 + 10 = 39$ |

# 1. Traveling Salesman Problem (3)




| No. | Traveling route | Weight |
|---|---|---|
| 1. | $a \to b \to c \to d \to a$ | $10 + 8 + 12 + 9 = 39$ |
| 2. | $a \to b \to d \to c \to a$ | $10 + 8 + 12 + 4 = 34$ |
| 3. | $a \to c \to b \to d \to a$ | $4 + 8 + 8 + 9 = 29$ |
| 4. | $a \to c \to d \to b \to a$ | $4 + 12 + 8 + 10 = 34$ |
| 5. | $a \to d \to b \to c \to a$ | $9 + 8 + 8 + 4 = 29$ |
| 6. | $a \to d \to c \to b \to a$ | $9 + 12 + 8 + 10 = 39$ |

Rute terpendek diberikan oleh:

- $a \to c \to b \to d \to a$, dengan bobot 29
- $a \to d \to b \to c \to a$, dengan bobot 29

# 1. Traveling Salesman Problem (4)

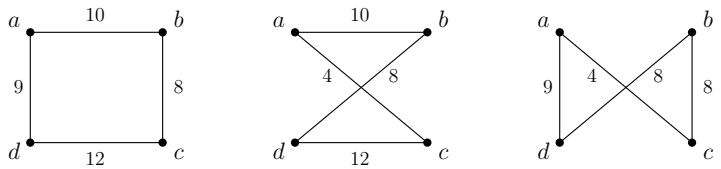
**Algoritma exhaustive search untuk solusi TSP**

1. Hitung semua sirkuit Hamilton dari graf lengkap *n*-simpul.
2. Evaluasi bobot setiap sirkuit Hamilton yang ditemukan pada langkah 1.
3. Pilih sirkuit Hamilton dengan bobot minimum.

Diskusikan bagaimana cara menghitung kompleksitasnya?

# Kompleksitas waktu of exhaustive search of TSP

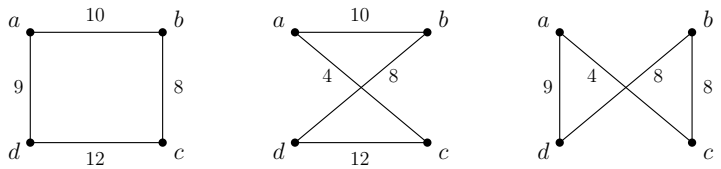
- Up to *shifting*, jumlah sirkuit Hamilton yang berbeda pada $n$ simpul adalah: $\frac{(n-1)!}{2}$ (gunakan "rumus permutasi siklik" (`https://www.wikiwand.com/en/Cyclic_permutation`), dan perhatikan bahwa kumpulan solusi dapat dikelompokkan menjadi pasangan-pasangan di mana yang satu merupakan cerminan dari yang lain).




- Jadi, untuk menyelesaikan TSP dengan pencarian menyeluruh, maka kita harus menghitung $\frac{(n-1)!}{2}$ sirkuit Hamilton, menghitung bobotnya, dan memilih sirkuit yang memiliki bobot minimum.
- Untuk menghitung bobot dari sebuah sirkuit, kita memerlukan waktu $\mathcal{O}(n)$.
- Oleh karena itu, kompleksitasnya adalah: $\frac{(n-1)!}{2} \cdot \mathcal{O}(n) \in \mathcal{O}(n \cdot n!)$ (tidak cukup baik).

# Kompleksitas waktu of exhaustive search of TSP

- Up to *shifting*, jumlah sirkuit Hamilton yang berbeda pada $n$ simpul adalah: $\frac{(n-1)!}{2}$ (gunakan "rumus permutasi siklik" (`https://www.wikiwand.com/en/Cyclic_permutation`), dan perhatikan bahwa kumpulan solusi dapat dikelompokkan menjadi pasangan-pasangan di mana yang satu merupakan cerminan dari yang lain).




- Jadi, untuk menyelesaikan TSP dengan pencarian menyeluruh, maka kita harus menghitung $\frac{(n-1)!}{2}$ sirkuit Hamilton, menghitung bobotnya, dan memilih sirkuit yang memiliki bobot minimum.
- Untuk menghitung bobot dari sebuah sirkuit, kita memerlukan waktu $\mathcal{O}(n)$.
- Oleh karena itu, kompleksitasnya adalah: $\frac{(n-1)!}{2} \cdot \mathcal{O}(n) \in \mathcal{O}(n \cdot n!)$ (tidak cukup baik).

# Kompleksitas waktu of exhaustive search of TSP

Latihan. Diberikan $n = 20$, jika waktu untuk mengevaluasi satu sirkuit Hamilton adalah 1 detik, berapa banyak waktu yang diperlukan untuk mendapatkan sirkuit Hamilton dengan bobot minimum!

# Kompleksitas waktu of exhaustive search of TSP

Latihan. Diberikan $n = 20$, jika waktu untuk mengevaluasi satu sirkuit Hamilton adalah 1 detik, berapa banyak waktu yang diperlukan untuk mendapatkan sirkuit Hamilton dengan bobot minimum!

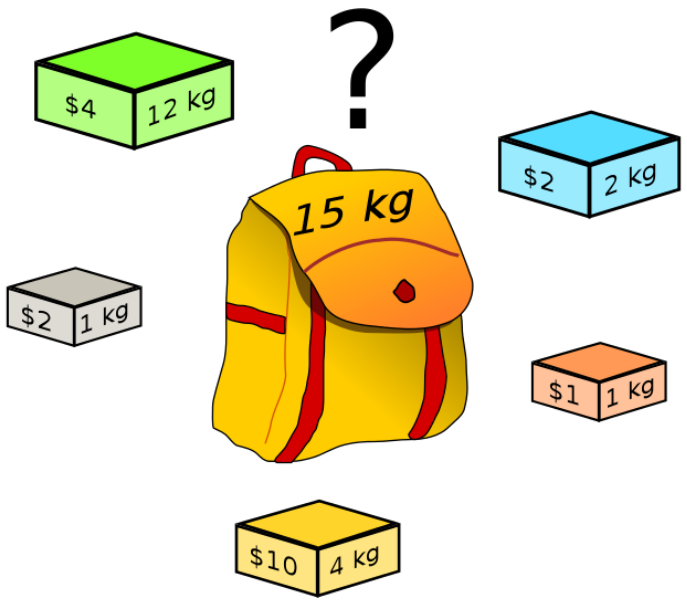
$\approx 1,541,911,905,814$ **tahun**

> *Ini karena kompleksitas waktu algoritma exhaustive search yang digunakan sangat besar, yaitu* $\mathcal{O}(n \cdot n!)$.

# 4.2. *Exhaustive search* untuk Permasalahan *1/0 Knapsack*

# 2. Permasalahan 1/0 Knapsack (1) [page 143]

Diberikan $n$ objek dan ransel berkapasitas $K$. Setiap objek $i$ memiliki bobot $w_i$ dan untung $p_i$. Tentukan cara menyeleksi objek ke dalam ransel agar keuntungan maksimal. Berat total benda tidak boleh melebihi kapasitas ransel.




Catatan. 1/0 knapsack artinya suatu objek dapat dimasukkan ke dalam knapsack (1) atau tidak termasuk (0).

# 2. Permasalahan 1/0 Knapsack (2): algoritma

**Exhaustive search untuk 1/0 knapsack problem:**

1. Tentukan semua himpunan bagian dari himpunan pada elemen $n$
2. Evaluasi keuntungan dari setiap subset pada langkah 1
3. Pilih subset yang memberikan keuntungan maksimum tetapi bobotnya tidak melebihi kapasitas ransel

# 2. Permasalahan 1/0 Knapsack (3): contoh

Diberikan empat objek dan ransel berkapasitas $K = 16$. Karakteristik dari setiap objek dirangkum dalam tabel berikut:

| **Object** | **Weight** | **Profit** |
|---|---|---|
| 1 | 2 | 20 |
| 2 | 5 | 30 |
| 3 | 10 | 50 |
| 4 | 5 | 10 |

# 2. Permasalahan 1/0 Knapsack (4): contoh

| Subset | Weight | Profit |
|---|---|---|
| $\{\}$ | 0 | 0 |
| $\{1\}$ | 2 | 20 |
| $\{2\}$ | 5 | 30 |
| $\{3\}$ | 10 | 50 |
| $\{4\}$ | 5 | 10 |
| $\{1,2\}$ | 7 | 50 |
| $\{1,3\}$ | 12 | 70 |
| $\{1,4\}$ | 7 | 30 |

| Subset | Weight | Profit |
|---|---|---|
| $\{2,3\}$ | 15 | 80 |
| $\{2,4\}$ | 10 | 40 |
| $\{3,4\}$ | 15 | 60 |
| $\{1,2,3\}$ | 17 | not feasible |
| $\{1,2,4\}$ | 12 | 60 |
| $\{1,3,4\}$ | 17 | not feasible |
| $\{2,3,4\}$ | 20 | not feasible |
| $\{1,2,3,4\}$ | 22 | not feasible |

Solusi optimal diberikan oleh subset $\{2,3\}$ dengan profit 80. Jadi solusi dari soal tersebut adalah $X = \{0,1,1,0\}$ (objek 1 dan 4 tidak diambil, dan objek 2 dan 3 diambil).

Catatan. Kandidat solusi "tidak layak", karena berat total melebihi kapasitas knapsack.

# 2. Permasalahan 1/0 Knapsack (5): Analisis kompleksitas waktu

**Kompleksitas waktu:**

- Jumlah himpunan bagian dari sekumpulan elemen $n$ adalah: $2^n$.
- Waktu untuk menghitung bobot total setiap sub-himpunan adalah: $\mathcal{O}(n)$.
- Jadi, kompleksitas *exhaustive search* untuk masalah 1/0 Knapsack adalah: $\mathcal{O}(n \cdot 2^n)$ (kompleksitas eksponensial).

# 2. Permasalahan 1/0 Knapsack (6): formulasi matematis

Kita juga dapat merepresentasikan masalah pengoptimalan secara matematis.

Tulis solusinya sebagai $X = \{x_1, x_2, \ldots, x_n\}$ dimana:

- $x_i = 1$, jika objek ke-$i$ dipilih
- $x_i = 0$ sebaliknya

Rumusan matematis Permasalahan 1/0 Knapsack:

**Definisi (math formulation of 1/0 knapsack)**

$$\textbf{Maximize } F = \sum_{i=1}^{n} p_i x_i$$

$$\textbf{subject to } \sum_{i=1}^{n} w_i x_i \leq K$$
$$\textit{and } x_i = 0 \textit{ or } x_i = 1, \textit{ for } i = 1, 2, \ldots, n$$

- Maksimalkan $F$ : fungsi tujuan
- terhadap ... : fungsi kendala (batasan)

# 3. Pencarian *exhaustive* pada bidang kriptografi

*Exhaustive search* digunakan dalam kriptografi sebagai teknik yang digunakan oleh penyerang untuk menemukan kunci dekripsi dengan mencoba semua kunci yang mungkin, dikenal sebagai exhaustive key search attack atau brute force attack.

## Example

Panjang kunci enkripsi pada algoritma DES (Data Encryption Standard) adalah 64 bit.

- Dari 64 bit tersebut, hanya 56 bit yang digunakan, sedangkan 8 bit lainnya digunakan sebagai pengecekan paritas.
- Jumlah kombinasi kunci adalah $2^{56} = 72,057,594,037,927,936$
- Artinya jika waktu yang diperlukan untuk mencoba satu kombinasi adalah 1 detik, maka untuk mencoba semua kombinasi membutuhkan waktu 2.284.931.317 tahun.

# **Bagian 5:** Latihan

# Assignment problem

# Latihan: 1. Masalah penugasan (*assignment problem*) (1) [page 145]

Diberikan $n$ staf dan $n$ tugas. Setiap staff diberikan sebuah tugas. Staff ($s_i$) ditugaskan ke tugas ($t_j$) dengan biaya $c(i,j)$. Rancang algoritma brute force untuk menetapkan tugas sedemikian rupa sehingga total biaya $\sum c(i,j)$ diminimalkan. Instance dari masalah direpresentasikan dalam matriks berikut.

Contoh.

**Matriks biaya:**

$$C = \begin{bmatrix} \textit{task 1} & \textit{task 2} & \textit{task 3} & \textit{task 4} \\ 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix} \begin{matrix} \\ \textit{staff a} \\ \textit{staff b} \\ \textit{staff c} \\ \textit{staff d} \end{matrix}$$

# Latihan: 1. Masalah penugasan (*assignment problem*) (1)

Catatan. Sebuah instance dari masalah penugasan (*masalah penugasan*) secara lengkap ditentukan oleh matriks biayanya.

Pertanyaan: Bagaimana Anda menemukan solusi pada matriks ini?

# Latihan: 1. Masalah penugasan (*assignment problem*) (1)

Catatan. Sebuah instance dari masalah penugasan (*masalah penugasan*) secara lengkap ditentukan oleh matriks biayanya.

Pertanyaan: Bagaimana Anda menemukan solusi pada matriks ini?

Beberapa iterasi pertama untuk memecahkan contoh kecil dari masalah penugasan (*assignment problem*) dengan exhaustive search.

$$C = \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix}$$

$$\begin{array}{ll} \langle 1,2,3,4 \rangle & \text{cost} = 9+4+1+4 = 18 \\ \langle 1,2,4,3 \rangle & \text{cost} = 9+4+8+9 = 30 \\ \langle 1,3,2,4 \rangle & \text{cost} = 9+3+8+4 = 24 \\ \langle 1,3,4,2 \rangle & \text{cost} = 9+3+8+6 = 26 \\ \langle 1,4,2,3 \rangle & \text{cost} = 9+7+8+9 = 33 \\ \langle 1,4,3,2 \rangle & \text{cost} = 9+7+1+6 = 23 \\ \vdots & \vdots \end{array}$$

Catatan. $\langle k, l, m, n \rangle$ berarti entri $c_{1,k}, c_{2,l}, c_{3,m}, c_{4,n}$.

# Latihan: 1. Masalah penugasan (*assignment problem*) (2)

**Kompleksitas**

- Banyaknya pilihan: $n \cdot (n-1) \cdot (n-2) \cdot \cdots \cdot 1$
- Kompleksitas: $\mathcal{O}(n^2)$

# Partition problem

# Latihan: 2. Masalah partisi (1)

Diberikan sebuah himpunan yang terdiri dari $n$ bilangan bulat positif. Partisi himpunan tersebut menjadi dua himpunan terpisah sedemikian rupa sehingga jumlah kedua subhimpunan itu sama. Rancang algoritma *exhaustive search* untuk masalah ini.

Contoh. Diberikan $n = 6$, dan himpunan bilangan: $3, 8, 4, 6, 1, 2$. Himpunan bilangan tersebut dapat dibagi menjadi dua subhimpunan yaitu: $\{3, 8, 1\}$ dan $\{4, 6, 2\}$, dimana jumlah masing-masingnya adalah 12.

Pertanyaan. Bagaimana menyelesaikan permasalahan ini?

# Latihan: 2. Masalah partisi (1)

## Algoritma

**Input:** himpunan $S$

**Output:** subset $A \subseteq S$ memuaskan $\text{sum}(A) = \frac{\text{sum}(S)}{2}$

1. Hitunglah semua himpunan bagian yang mungkin dari $S$;
2. Untuk setiap subset $A \subseteq S$, periksa apakah $\text{sum}(A) = \frac{\text{sum}(S)}{2}$;

# Latihan: 2. Masalah partisi (1)

## Algoritma

**Input:** himpunan $S$

**Output:** subset $A \subseteq S$ memuaskan $\text{sum}(A) = \frac{\text{sum}(S)}{2}$

1. Hitunglah semua himpunan bagian yang mungkin dari $S$;
2. Untuk setiap subset $A \subseteq S$, periksa apakah $\text{sum}(A) = \frac{\text{sum}(S)}{2}$;

Kompleksitas waktu: $\mathcal{O}(2^n)$
(karena satu set elemen $n$ memiliki subset $2^n$).

# Magic square

# Latihan: 3. Persegi ajaib (bagian 1)

Persegi ajaib adalah susunan $n$ angka dari 1 sampai $n^2$ dalam kuadrat berukuran $n \times n$ sehingga jumlah setiap kolom, baris, dan diagonal adalah sama. Rancang algoritma *exhaustive search* untuk membangun persegi ajaib berukuran $n$.

| | | |
|---|---|---|
| 4 | 9 | 2 |
| 3 | 5 | 7 |
| 8 | 1 | 6 |

# Latihan: 3. Persegi ajaib (bagian 2)

## Algoritma

**Input:** $(1, 2, 3, \ldots n^2)$

**Output:** sebuah persegi ajaib berukuran $n \times n$

1. Hitung semua kuadrat yang mungkin;
2. Untuk masing-masing, periksa apakah itu adalah persegi ajaib (dengan memeriksa apakah jumlah setiap baris, kolom, dan diagonal sama).

# Latihan: 3. Persegi ajaib (bagian 2)

## Algoritma

**Input:** $(1, 2, 3, \ldots n^2)$

**Output:** sebuah persegi ajaib berukuran $n \times n$

1. Hitung semua kuadrat yang mungkin;
2. Untuk masing-masing, periksa apakah itu adalah persegi ajaib (dengan memeriksa apakah jumlah setiap baris, kolom, dan diagonal sama).

Kompleksitas: $\mathcal{O}(n!)$ karena ada $n!$ kemungkinan persegi

# Latihan: menuliskan dalam bentuk pseudocode

**Tugas**: Tulis pseudocode untuk ketiga latihan tersebut!

# **Bagian 6:** Teknik heuristik

# Teknik heuristik (1)

## Bahan bacaan:

`https://www.wikiwand.com/en/Heuristic_(computer_science)`

Heuristik adalah teknik yang dirancang untuk memecahkan masalah lebih cepat ketika metode klasik terlalu lambat atau untuk menemukan solusi perkiraan ketika metode klasik gagal menemukan solusi eksak.

- Tujuan dari heuristik adalah untuk menghasilkan solusi dalam kerangka waktu yang masuk akal yang cukup baik untuk memecahkan masalah.
- Heuristik menggunakan *tebakan* atau *intuisi* yang tidak dapat dibuktikan secara matematis.
- Tidak selalu memberikan solusi optimal.
- Heuristik yang baik dapat sangat mengurangi waktu untuk menyelesaikan masalah dengan menghilangkan kandidat solusi yang tidak perlu.
- Tidak ada jaminan bahwa heuristik dapat memecahkan masalah, tetapi ini bekerja berkali-kali dan seringkali lebih cepat daripada *exhaustive search*.

# Teknik heuristik (2)

Teknik heuristik dapat digunakan untuk mengurangi jumlah kandidat yang mungkin dari solusi masalah.

## Example

Dalam soal anagram, untuk bahasa Inggris kita dapat menggunakan aturan bahwa huruf "c" dan "h" sering muncul berurutan dalam kata bahasa Inggris. Jadi kita hanya dapat mempertimbangkan permutasi huruf di mana "ch" muncul bersamaan.

**Contoh.**

- march $\rightarrow$ charm
- chapter $\rightarrow$ patcher, repatch

# Teknik heuristik (3)

## Example

Untuk menyelesaikan masalah *magic square* dengan exhaustive search, kita harus memeriksa $9! = 362,880$ solusi yang mungkin, lalu periksa apakah untuk masing-masing solusi, jumlah setiap kolom, baris, dan diagonalnya sama.

Dengan Teknik Heuristik, untuk setiap solusi, kita dapat memeriksa apakah kolom pertama memiliki **sum = 15**. Jika ya, kita periksa kolom/baris berikutnya. Jika tidak, kita berhenti, dan memeriksa permutasi lainnya.

# Teknik heuristik (4)

**Trade-off:** Kapan harus menggunakan Teknik Heuristik?

Hal-hal yang menjadi pertimbangan adalah sebagai berikut.

- Optimality: Ketika ada beberapa solusi untuk masalah tertentu, apakah heuristik menjamin bahwa solusi terbaik akan ditemukan? Apakah sebenarnya perlu untuk menemukan solusi terbaik?
- Completeness: Ketika ada beberapa solusi untuk masalah tertentu, dapatkah heuristik menemukan semuanya? Apakah kita benar-benar membutuhkan semua solusi? Banyak teknik heuristik hanya dimaksudkan untuk menemukan satu solusi.
- Accuracy and precision: Bisakah heuristik memberikan interval kepercayaan untuk solusi yang diperoleh? Apakah *gap* kesalahan (antara solusi heuristik dengan solusi sebenarnya) pada solusi masih dalam batas yang ditolerir?
- Execution time: Apakah tekik yang digunakan merupakan teknik heuristik terbaik untuk menyelesaikan masalah tersebut, ditinjau dari segi kompleksitas waktu?

*to be continued...*