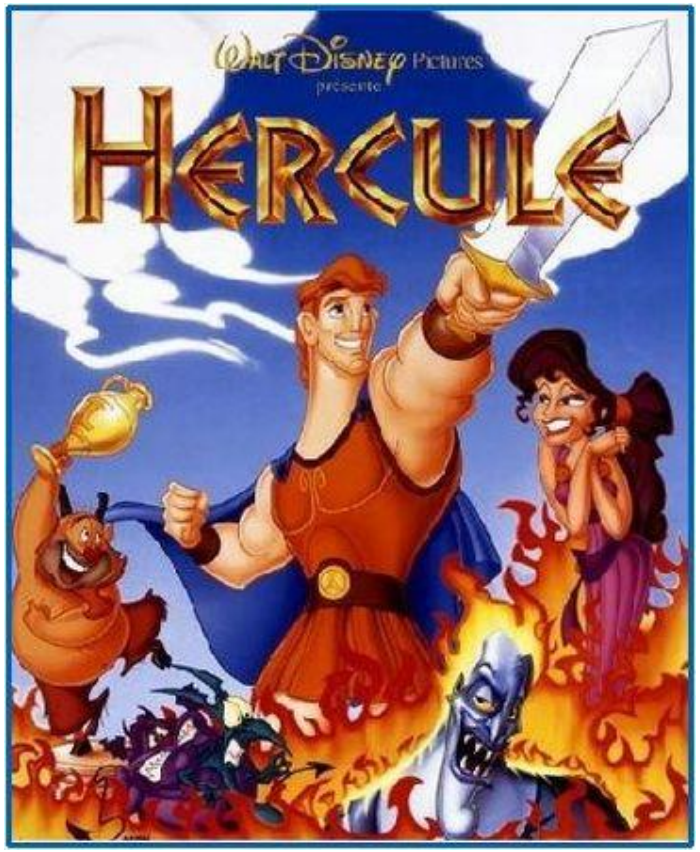
Bahan Kuliah
IF2211 Strategi Algoritma

Algoritma *Brute Force*

(Bagian 2)



Oleh: Rinaldi Munir

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika, ITB, 2025

Latihan

(yang diselesaikan secara *exhaustive search*)

1. (**Persoalan Penugasan**) Misalkan terdapat *n* orang dan *n* buah pekerjaan (*job*). Setiap orang akan di-*assign* dengan sebuah pekerjaan. Penugasan orang ke-*i* dengan pekerjaan ke-*j* membutuhkan biaya sebesar *c*(*i*, *j*). Bagaimana melakukan penugasan sehingga total biaya penugasan adalah seminimal mungkin? Misalkan instansiasi persoalan dinyatakan sebagai matriks *C* sebagai berikut

$$C = \begin{bmatrix} Job\,1 & Job\,2 & Job\,3 & Job\,4 & \\ 9 & 2 & 7 & 8 & \text{Orang } a \\ 6 & 4 & 3 & 7 & \text{Orang } b \\ 5 & 8 & 1 & 4 & \text{Orang } c \\ 7 & 6 & 9 & 4 & \text{Orang } d \end{bmatrix}$$

2. (**Persoalan partisi**). 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.

   Rancang algoritma *exhaustive search* untuk masalah ini. Cobalah mengurangi jumlah himpunan bagian yang perlu dibangkitkan.

3. (**Bujursangkar ajaib**). Bujursangkar ajaib (*magic square*) adalah pengaturan *n* buah bilangan dari 1 hingga n^2 di dalam bujursangkar yang berukuran *n* x *n* sedemikian sehingga jumlah nilai setiap kolom,baris, dan diagonal sama. Rancanglah algoritma *exhaustive search* untuk membangkitkan bujursangkar ajaib orde *n*.

4	9	2
3	5	7
8	1	6

Exhaustive Search di dalam Kriptografi

- Di dalam kriptografi, *exhaustive search* merupakan teknik yang digunakan penyerang untuk menemukan kunci dekripsi dengan cara mencoba semua kemungkinan kunci.

  Serangan semacam ini dikenal dengan nama *exhaustive key search attack* atau *brute force attack*.

- Serangan yang sama juga digunakan dalam menemukan *password*, PIN, dan kode rahasia lainnya.

Contoh 5: Panjang kunci enkripsi di dalam algoritma *DES* (*Data Encryption Standard*) adalah 64 bit.

- Dari 64 bit tersebut, hanya 56 bit yang digunakan (8 bit paritas lainnya tidak dipakai).

- Jumlah kombinasi kunci yang harus dicoba (dievaluasi) oleh pihak lawan adalah sebanyak

$$(2)(2)(2)(2)(2) \ldots (2)(2) = 2^{56} = 72.057.594.037.927.936$$

- Jika untuk percobaan dengan satu kunci memerlukan waktu 1 detik, maka untuk jumlah kunci sebanyak itu diperlukan waktu komputasi kurang lebih selama 2.284.931.317 tahun!

- Algoritma *exhaustive search* tidak sangkil sebagaimana ciri algoritma *brute force* pada umumnya, karena membutuhkan volume komputasi yang besar untuk menyelesaikan persoalan dengan meningkatnya ukuran masukan.

- Namun, nilai plusnya terletak pada keberhasilannya yang selalu menemukan solusi (jika diberikan waktu yang mencukupi).

Menemukan *password* file ZIP secara *exhaustive search* dengan *dictionary attack*

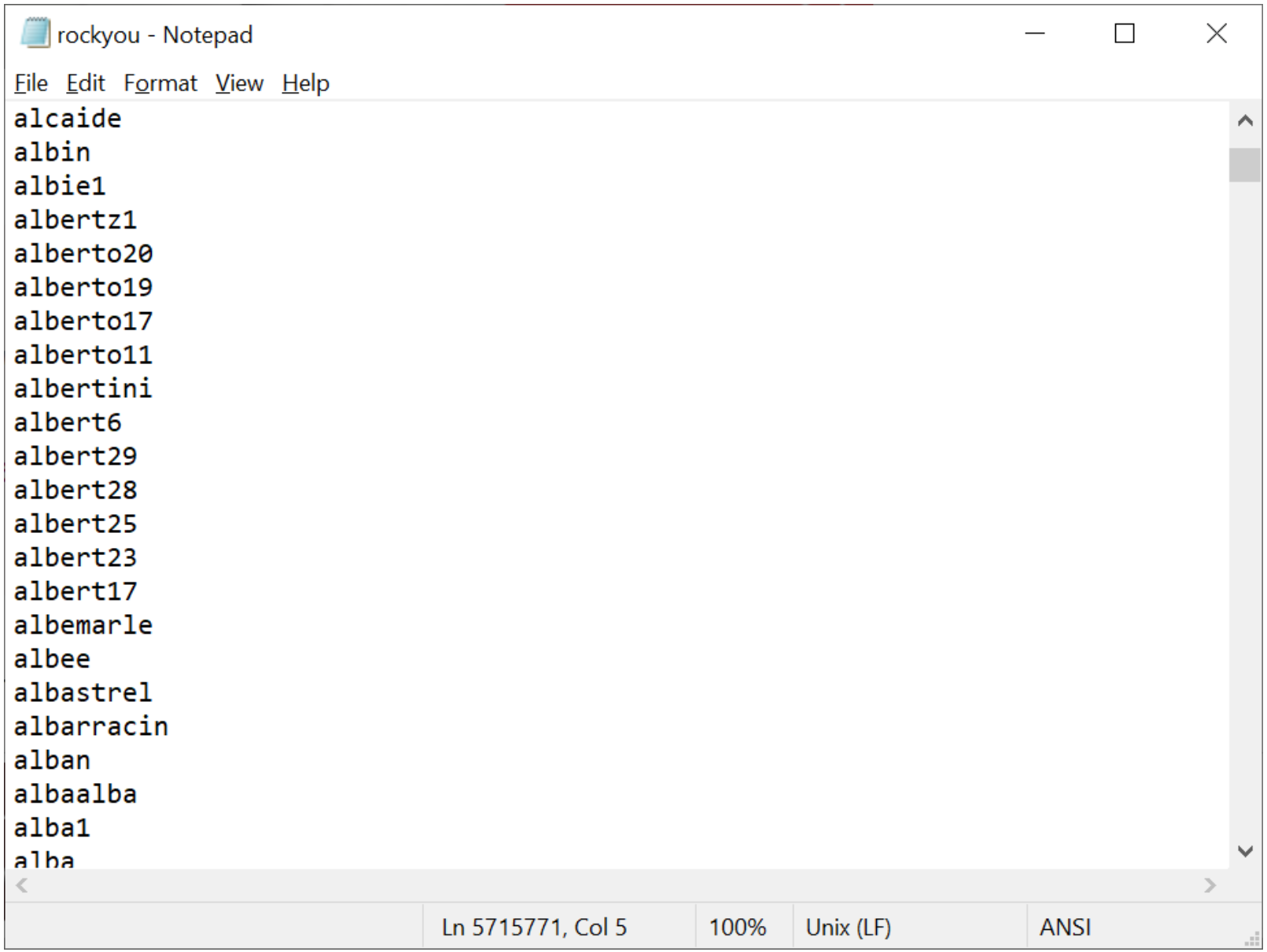
- Sumber: https://www.thepythoncode.com/article/crack-zip-file-password-in-python
- *Dictionary attack*: mencoba berbagai kemungkinan *password*/kunci dengan menggunakan kamus yang berisi semua kemungkinan *password* yang dibentuk dari kombinasi semua informasi penting yang terkait dengan pemilik data rahasia
- Contoh: Willy adalah mahasiswa IF 2023 yang berasal dari Palembang. Dia tinggal di Jalan Bengawan No 34. Orangtuanya Bernama Iqbal Sanjaya, ibunya bernama Fatmawati. Willy lahir tanggal 26 Mei 2003.
- Kombinasi password yang mungkin:
  - bengawan2003
  - balfat2020
  - PalMei26
  - palemb34
  - wil26503
  - dll

- Program exhaustive search / brute force dalam Python
  (Sumber: https://www.thepythoncode.com/article/crack-zip-file-password-in-python)

```
import zipfile
from tqdm import tqdm
 # the password list path you want to use, must be available in the current directory
wordlist = "rockyou.txt"
# the zip file you want to crack its password
zip_file = "secret.zip"
# initialize the Zip File object
zip_file = zipfile.ZipFile(zip_file)
# count the number of words in this wordlist
n_words = len(list(open(wordlist, "rb")))
# print the total number of passwords
print("Total passwords to test:", n_words)
```

```
 with open(wordlist, "rb") as wordlist:
for word in tqdm(wordlist, total=n_words, unit="word"):
    try:
        zip_file.extractall(pwd=word.strip())
    except:
        continue
    else:
        print("[+] Password found:", word.decode().strip())
        exit(0)
print("[!] Password not found, try other wordlist.")
```

Contoh semua kemungkinan password di dalam file rockyou.txt



Teknik Heuristik

- *Algoritma exhaustive search* dapat diperbaiki kinerjanya sehingga tidak perlu melakukan pencarian solusi dengan mengeksplorasi semua kemungkinan solusi.

- Salah satu teknik yang digunakan untuk mempercepat pencarian solusi di dalam *exhaustive search* adalah teknik **heuristik** (*heuristic*). Terkadang disebut juga **fungsi heuristik**.

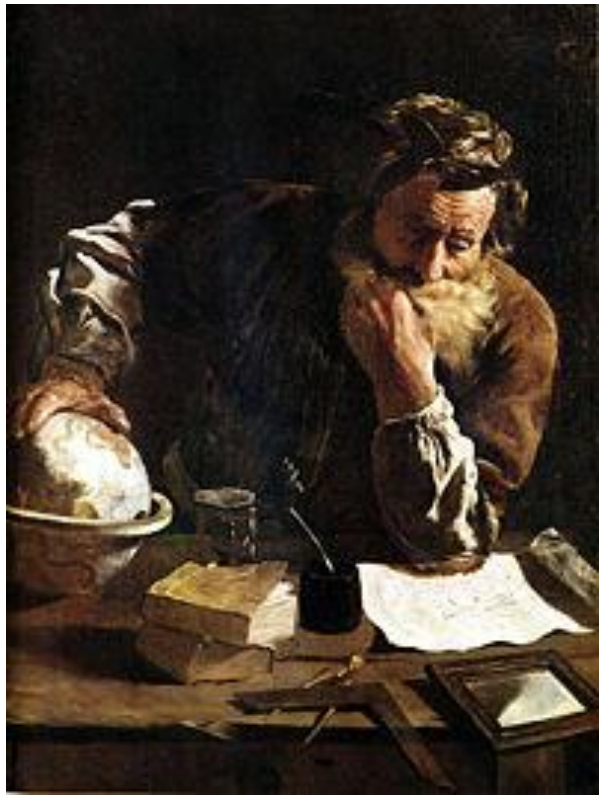
- Dalam *exhaustive search*, teknik heuristik digunakan untuk mengeliminasi beberapa kemungkinan solusi tanpa harus mengeksplorasi seluruh kemungkinan solusi secara penuh.

- Heuristik merupakan teknik yang dirancang untuk memecahkan persoalan dengan mengabaikan apakah teknik tersebut terbukti benar secara matematis

- Sebab teknik ini menggunakan pendekatan yang tidak formal, misalnya berdasarkan penilaian intuitif, terkaan, atau akal sehat.

- Contoh: program antivirus menggunakan pola-pola heuristik untuk mengidentifikasi dokumen yang terkena virus atau *malware*.

Asal Mula Kata Heuristik

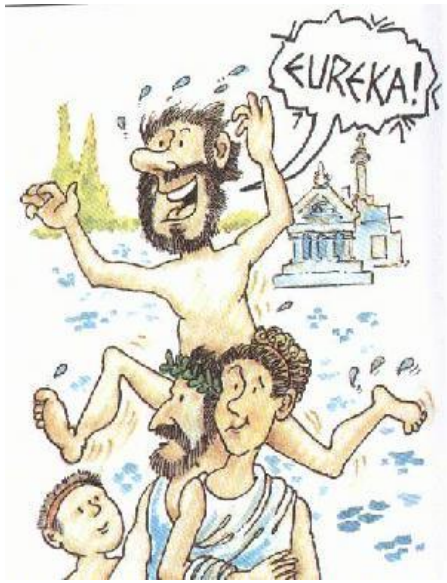


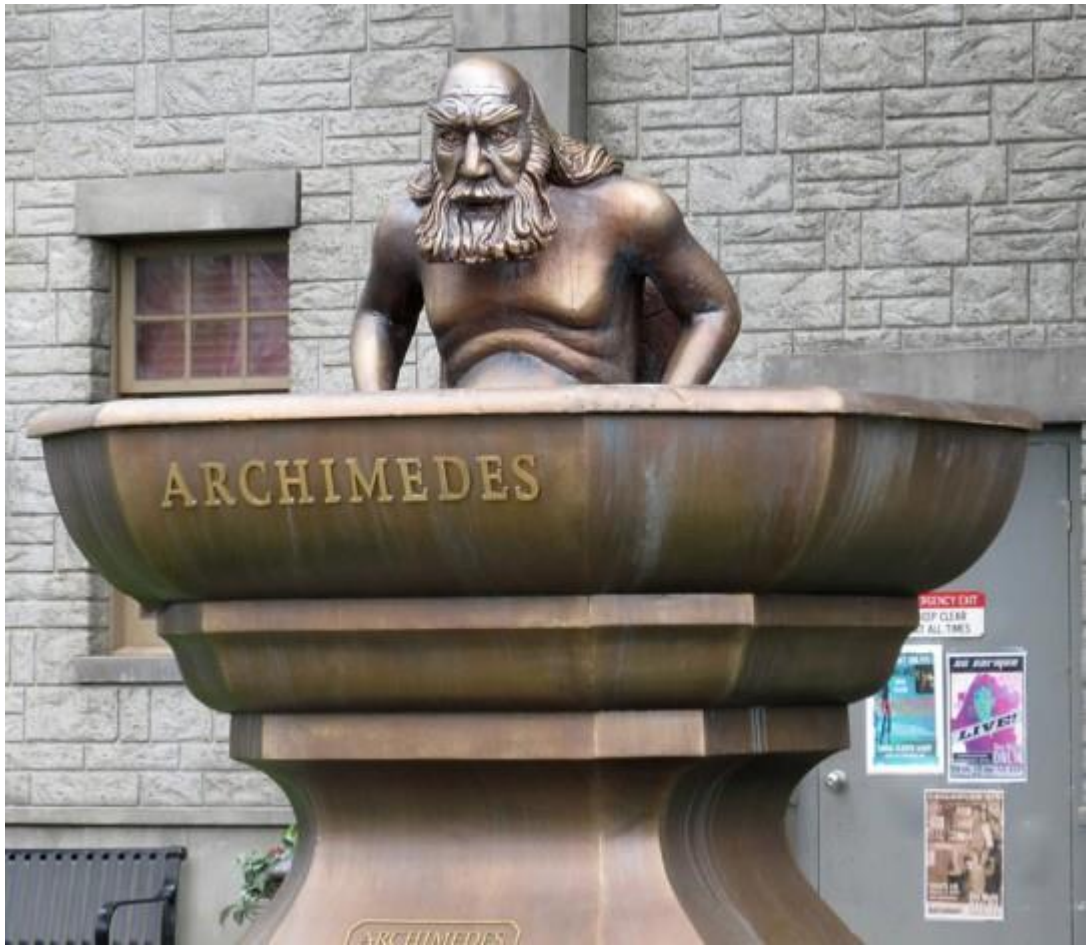
- Heuristik adalah seni dan ilmu menemukan (*art and science of discovery*).

- Kata heuristik diturunkan dari Bahasa Yunani yaitu “*eureka*” yang berarti “menemukan” (*to find* atau *to discover*).

- Matematikawan Yunani yang bernama Archimedes yang melontarkan kata "*heureka*", dari sinilah kita menemukan kata “*eureka*” yang berarti *“I have found it*.”


Eureka!


EUREKA!


ARCHIMEDES

- Heuristik mengacu pada teknik memecahkan persoalan berbasis pengalaman, dari proses pembelajaran, dan penemuan solusi meskipun tidak dijamin optimal.

- Heuristik berbeda dengan algoritma:
  - heuristik berlaku sebagai panduan (*guideline*),
  - sedangkan algoritma adalah urutan langkah-langkah penyelesaian persoalan.

- Teknik heuristik menggunakan terkaan, intuisi, dan *common sense*. Secara matematis tidak dapat dibuktikan, namun sangat berguna.

- Teknik heuristik mungkin tidak selalu memberikan hasil optimal, tetapi secara ekstrim ia berguna pada penyelesaian persoalan.

- Heuristik yang bagus dapat secara dramatis mengurangi waktu yang dibutuhkan untuk memecahkan persoalan dengan cara mengeliminir kebutuhan untuk mempertimbangkan kemungkinan solusi yang tidak perlu.

- Heuristik tidak menjamin selalu dapat memecahkan persoalan, tetapi seringkali memecahkan persoalan dengan cukup baik untuk kebanyakan persoalan, dan seringkali pula lebih cepat daripada pencarian solusi secara *exhaustive search*.

- Sudah sejak lama heuristik digunakan secara intensif di dalam bidang intelijensia buatan (*artificial intelligence*), misalnya di dalam metode hill climbing, best first search, algoritma A*, dan lain-lain.

Contoh penggunaan heuristik untuk mempercepat algoritma exhaustive search

Contoh 6: Persoalan *anagram*. *Anagram* adalah penukaran huruf dalam sebuah kata atau kalimat sehingga kata atau kalimat yang baru mempunyai arti lain.

Contoh-contoh *anagram* (semua contoh dalam Bahasa Inggris):

lived $\rightarrow$ *devil*

listen $\rightarrow$ *silent*

tea $\rightarrow$ *eat*

charm $\rightarrow$ *march*

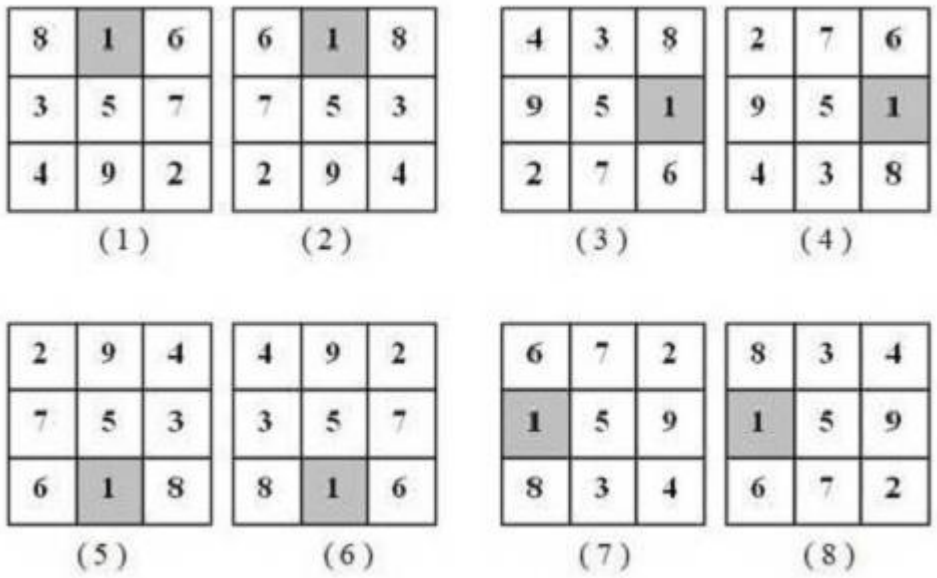
- Bila diselesaikan secara *exhaustive search*, kita harus mencari semua permutasi huruf-huruf pembentuk kata atau kalimat, lalu memerika apakah kata atau kalimat yang terbentuk mengandung arti (mengacu ke kamus).

- Teknik heuristik dapat digunakan untuk mengurangi jumlah pencarian solusi.

- Salah satu teknik heuristik yang digunakan misalnya membuat aturan bahwa dalam Bahasa Inggris huruf *c* dan *h* selalu digunakan berdampingan sebagai *ch* (lihat contoh *charm* dan *march*), sehingga kita hanya membuat permutasi huruf-huruf dengan *c* dan *h* berdampingan.

- Dengan demikian, semua permutasi dengan huruf *c* dan *h* tidak berdampingan ditolak dari pencarian $\rightarrow$ mengurangi volume komputasi

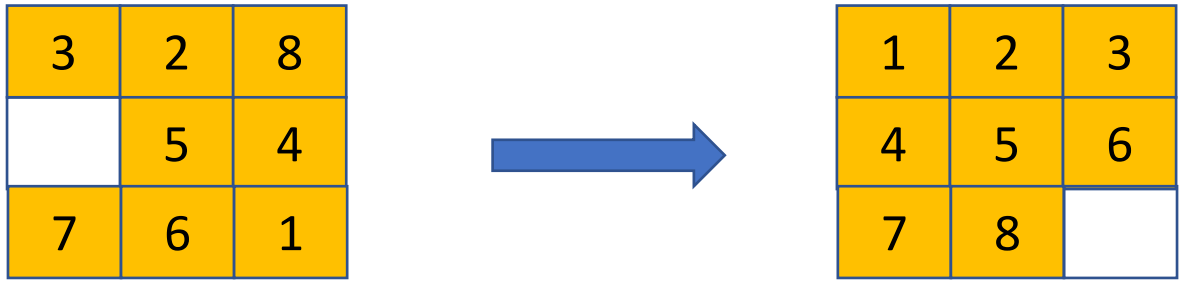
Contoh 7: Kubus ajaib 3 x 3 mempunyai 8 buah solusi sbb



Dengan *exhaustive search*, kita perlu memeriksa 9! = 362,880 susunan solusi yang mungkin, kemudian memeriksa apakah umlah setiap baris, setiap kolom, atau setiap diagonal selalu 15.

Teknik heuristik: pada setiap pembangkitan permutasi, periksa apakah nilai 3 buah angka pertama jumlahnya 15. Jika ya → teruskan, jika tidak → tolak

Contoh 8: Pada permainan 8-*puzzle*, fungsi heuristik digunakan untuk menentukan *cost* minimal dari suatu status ke status lainnya di dalam ruang pencarian.



Status awal Status akhir

Fungsi heuristik:

$h(n)$ = jumlah ubin yang tidak terdapat pada status akhir

= 5

= dibutuhkan *minimal* 5 pergerakan ubin untuk mencapai status akhir dari status awal

Latihan Soal *Brute Force* dan *Exhaustive Search*

1. Soal UTS 2014

- Diberikan sebuah larik (*array*) integer a_1, a_2, …, a_n. Anda diminta menemukan *sub-sequence* yang kontigu (berderetan) dari larik tersebut yang memiliki nilai maksimum. Nilai maksimum *sub-sequenc*e adalah nol jika semua elemen larik adalah negatif. Sebagai contoh instansiasi: larik [–2 , 11, –4 , 13, –5 , 2, –1, 3] memiliki nilai maksimum *sub-sequence* kontigu adalah 20, yaitu dari elemen ke-2 hingga elemen ke-4 (yaitu [11, –4 , 13]).

- Jika diselesaikan dengan algoritma *brute force* bagaimana caranya? Berapa kompleksitas algoritma *brute force* tersebut dalam notasi O-besar?

Jawaban:

- **Alternatif 1** (*Exhaustive Search*)

  - Nyatakan larik sebagai sebuah himpunan

  - Tuliskan semua upa-himpunan (*sub-set*) dari himpunan tersebut, lalu sesuaikan urutan elemen di dalam upa-himpunan dengan urutan elemen di dalam larik.

    Contoh *sub-set* dari [–2 , 11, –4 , 13, –5 , 2, –1, 3] $\rightarrow$ {13, –4, 11} $\rightarrow$ [11, –4, 13]

  - Buang upa-himpunan yang elemen-elemennya tidak berurutan.

    Contoh *sub-set* dari [–2 , 11, –4 , 13, –5 , 2, –1, 3] $\rightarrow$ {11, 2, 3} $\rightarrow$ dibuang

  - Hitung jumlah nilai di dalam setiap upa-himpunan.

  - Pilih upa-himpunan yang mempunyai nilai maksimum.

Jumlah semua upa-himpunan adalah 2^n, menghitung jumlah nilai di dalam upa-himpunan adalah O(n), maka kompleksitas algoritmanya adalah O(n. 2^n).

- **Alternatif 2** (*Exhaustive Search*), lebih baik

  - Tuliskan semua *sub-sequence* dengan 1 elemen (ada *n* buah), *sub-sequence* dengan 2 elemen (ada $n - 1$ buah), dan seterusnya hingga *sub-sequence* dengan *n* elemen (1 buah). Seluruhnya ada

    $$n + (n - 1) + (n - 2) + \ldots + 2 + 1 = n(n + 1)/2 \quad \textit{sub-sequence}.$$

    Contoh: [8 , –6, 4 , 3, –5] $\rightarrow n = 5$

    Sub-sequence dengan 1 elemen: [8] , [–6], [4], [3] , [–5] $\rightarrow$ 5 *sub-sequence*

    Sub-sequence dengan 2 elemen: [8, –6], [–6, 4], [4, 3], [3, –5] $\rightarrow$ 4 *sub-sequence*

    Sub-sequence dengan 3 elemen: [8, –6, 4], [–6, 4, 3], [4, 3, –5] $\rightarrow$ 3 *sub-sequence*

    dst…

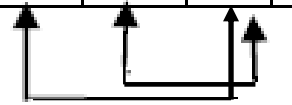
  - Hitung jumlah nilai pada setiap *sub-sequence*
  - Pilih *sub-sequence* yang jumlahnya maksimum

Menghitung jumlah nilai di dalam *sub-sequence* adalah O(*n*), maka kompleksitas algoritmanya adalah $O(n . n(n + 1)/2) = O(n^3)$.

2. Soal UTS tahun 2019

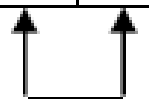
(*Inversion problem*) *Netflix* menggunakan sistem rekomendasi untuk merekomendasikan film yang anda sukai. *Netflix* mencoba mencocokkan film kesukaanmu dengan film lainnya. Sistem rekomendasi tersebut adalah sbb: Misalkan kamu me-rangking *n* buah film. Selanjutnya, *Netflix* memeriksa basisdatanya untuk mencari orang dengan kesukaan film yang mirip. Ukuran kemiripan yang digunakan adalah jumlah inversi antara kedua rangking. Misalkan ranking kamu adalah 1, 2, 3, …, *n*, sedangkan rangking orang lain adalah a_1, a_2, …, a_n. Film *i* dan film *j* disebut inversi jika $i < j$ tetapi $a_i > a_j$. Contoh untuk film A, B, C, D, dan E:

Film	A	B	C	D	E
Ranking saya	1	2	3	4	5
Ranking X	1	3	4	2	5



Inversi: (3, 2) dan (4, 2)

Film	A	B	C	D	E
Ranking saya	1	2	3	4	5
Ranking Y	1	2	4	3	5



Inversi (4, 3)

Karena jumlah inversi dengan Y lebih sedikit daripada X, maka kesukaan saya lebih mirip dengan Y.

Anda diminta menyelesaikan persoalan inversi sbb: Diberikan sebuah senarai *A* dengan *n* elemen. Hitunglah jumlah inversi di dalam senarai tersebut. Definisi inversi: Jika $i < j$ tetapi $A[i] > A[j]$ maka pasangan $(A[i], A[j])$ disebut inversi. Contoh: A = [1, 9, 6, 4, 5], maka jumlah inversi adalah 5, yaitu pasangan (9, 6), (9, 4, (9, 5), (6, 4), dan (6, 5).

(a) Hitung jumlah inversi dan pasangan inversinya pada senarai A = [1, 5, 4, 8, 10, 2, 6, 9, 3, 7]. **(7,5)**

(b) Jika persoalan inversi diselesaikan dengan algoritma *Brute-Force*, bagaimana langkah-langkahnya? Jelaskan! Berapa jumlah perbandingan elemen yang dibutuhkan dan berapa kompleksitas algoritma dalam notasi *Big-Oh*? **(7,5)**

Jawaban:

(a) **A**da 17 buah inversi: (5, 4), (4, 2), (8, 2), (10, 2), (6, 3), (9, 3)
(5, 2), (4, 3), (8, 6), (10, 6), (9, 7) (5, 3), (8, 3), (10, 9), (10, 3), (8, 7), (10, 7)

(b) Mulai dari i = 1...n-1
Mulai dari k = i+1..n
jika A[i] > A[k] maka catat sebagai inversi

$$T(n) = (n-1) + (n-2) + \ldots + 2 + 1 = n(n-1)/2 = O(n^2)$$

Soal latihan tambahan Algoritma Brute Force

IF2211 Strategi Algoritma

2024

UTS 2022

Diberikan sebuah larik yang berisi elemen biner (0 atau 1). Elemen-elemen larik sudah **terurut** menaik (dari kecil ke besar). Kita akan menghitung jumlah bit 1 di dalam larik tersebut. Contoh: A = [0, 0, 0, 1, 1, 1, 1, 1], jumlah bit 1 adalah 5. Jika diselesaikan dengan algoritma brute force, jelaskan caranya atau langkah-langkahnya, lalu tentukan kompleksitas waktu asimptotiknya dalam notasi O besar.

Jawaban:

- Traversal larik sampai ketemu elemen 1 pertama. Karena larik sudah terurut, maka hitung jumlah 1 dari posisi tersebut sampai posisi terakhir.

- Kompleksitas waktu algoritmanya adalah O(n) karena:
  - traversal larik dengan mencari elemen 1 pertama: O(n)
  - menghitung banyaknya 1 dari posisi 1 pertama sampai terakhir: O(n)
  - total waktu: O(n) + O(n) = O(n)

UTS 2022

Seorang mahasiswa IF tingkat 4 mengalokasikan waktu m jam untuk mengerjakan proyek programming. Terdapat n proyek ($p_1..p_n$) yang mungkin diterima dengan mempertimbangkan estimasi waktu kerja ($t_1..t_n$) dalam jam, dan honor yang akan diterima ($s_1..s_n$) dalam ratusan ribu. Bantulah mahasiswa ini memilih proyek yang akan dikerjakan untuk memaksimumkan honor yang akan diterima.

Jika diselesaikan dengan exhaustive search, jelaskanlah langkah-langkah yang dilakukan di dalam exhaustive search. Lengkapilah penjelasan dengan jumlah kandidat solusi yang akan dievaluasi, dan tentukan kompleksitas waktu asimptotiknya dalam notasi O besar.

Jawaban:

Ini adalah persoalan knapsack problem. Langkah-langkah dalam exhaustive search:

i) enumerasi kandidat solusi dengan sistematis. Jumlah kandidat solusi adalah sebanyak subset dari n buah proyek: 2^n buah subset

ii) evaluasi setiap kandidat solusi dengan menghitung apakah total waktu yang dibutuhkan untuk mengerjakan proyek tidak melebihi m. Jika tidak melebihi m, hitung total honor. Kebutuhan waktu: O(n)

iii) umumkan solusi terbaik di akhir pencarian.

Kompleksitas waktu $O(n.2^n)$.

UTS 2021

- Misalkan terdapat sebuah larik $a[1..n]$ dengan n elemen bilangan bulat. Kita ingin menghitung $F = \sum_{i=1}^{n} i * a[i]$ sedemikian sehingga F bernilai maksimum. Jika diselesaikan secara *brute force/exhaustive search*, bagaimana caranya, dan berapa perkiraan kompleksitas waktunya (dalam notasi Big-Oh)?

Jawaban

- Cari semua permutasi elemen-elemen larik a[1], a[2], …, a[n], yaitu ada sebanyak n! susunan permutasi.
- Untuk setiap susunan, hitung $S = \sum_{i=1}^{n} i * a[i]$, lalu pilih susunan permutasi yang memiliki nilai S maksimum.

- Menghitung S = $\sum_{i=1}^{n} i * a[i]$ kompleksitsnya adalah O(n), karena ada n perkalian dan n pejumlahan.

- Jadi kompleksitas algoritma seluruhnya adalah O(n · n!).

UTS 2016

Misalkan anda diberikan sebuah larik bilangan bulat yang terurut. Setiap nilai muncul dua kali, kecuali sebuah nilai tertentu yang hanya muncul sekali. Tugas anda adalah mencari nilai integer yang muncul hanya sekali.

Contoh larik:

(i) 1, 1, 2, 2, **3**, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8

(ii) 10, 10, 17, 17, 18, 18, 19, 19, 21, 21, **23**

(iii) **1**,3, 3, 5, 5, 7, 7, 8, 8, 9, 9, 10, 10

(angka yang dicetak tebal adalah nilai yang hanya muncul sekali)

Jika diselesaikan dengan algoritma Brute Force, bagaimana caranya? (jawaban bukan dalam *pseudo-code*). Berapa kompleksitasnya dalam notasi O-besar?

TAMAT