

## 3.1 - Algoritma *Brute Force* (bagian 1)

[KOMS120403]

Desain dan Analisis Algoritma (2023/2024)

Dewi Sintiar

Prodi S1 Ilmu Komputer  
Universitas Pendidikan Ganesha

Week 3 (March 2024)

# Daftar isi

- Prinsip algoritma brute force
- Beberapa contoh teknik brute-force
  - 1 Finding max/min of array
  - 2 Sequential search
  - 3 Computing power
  - 4 Menghitung faktorial
  - 5 Square-matrix multiplication
  - 6 Prime-number test
  - 7 Evaluasi polinomial
  - 8 Masalah pasangan titik terdekat
  - 9 Pencocokan pola (*pattern matching*)
- Karakteristik algoritma brute force

# Tujuan pembelajaran

Anda diharapkan mampu untuk:

- 1 Menjelaskan prinsip dasar algoritma brute force
- 2 Menjelaskan penerapan algoritma brute force pada penyelesaian masalah algoritmik sederhana
- 3 Menjelaskan prinsip *exhaustive search* serta keterkaitannya dengan algoritma brute force

# Bagian 1: Konsep dasar algoritma brute force

# Algoritma brute force (1)

## Definisi (Algoritma Brute Force atau *Exhaustive Search*)

*Teknik pemecahan masalah yang menggunakan pendekatan langsung.*

*Solusinya ditemukan dengan **memeriksa setiap kemungkinan jawaban satu per satu**, untuk menentukan apakah hasilnya merupakan solusi dari permasalahan tersebut.*

Algoritma brute-force biasanya didasarkan pada:

- deskripsi masalah;
- definisi/konsep yang termuat pada masalah

**Karakteristik:** sederhana, pendekatan langsung (*direct*), cara yang jelas/terstruktur

## Algoritma brute force (2)

### Contoh sederhana algoritma brute force

- Diberikan bilangan bulat  $n$ , untuk menemukan semua pembagi dari  $n$ , seseorang dapat memeriksa apakah setiap bilangan bulat  $i \in [1, n]^*$  dapat membagi  $n$ .
- Diberikan *PIN* yang terdiri dari 4 digit angka (0 sampai dengan 9). Untuk mencari PIN yang benar dengan algoritma brute force, coba semua kemungkinan kombinasi satu per satu seperti 0001, 0002, 0003, 0004, dan seterusnya sampai kita mendapatkan PIN yang tepat (dalam hal ini, paling banyak terdapat 10.000 percobaan).

---

\*Notasi  $[1, n]$  artinya bilangan bulat dari 1 sampai dengan  $n$ .

# Bagian 2: Contoh penerapan algoritma *brute force*

## 2.1. Mencari nilai maksimum/minimum pada array



## 1. Mencari elemen max/min dari sebuah array

**Masalah.** Diberikan array  $n$  bilangan bulat  $(a_1, a_2, \dots, a_n)$ . Kita ingin menemukan nilai maksimum pada array.

**Pendekatan brute-force:** untuk menemukan nilai maksimum, bandingkan setiap elemen dari  $a_1$  hingga  $a_n$ .

---

### Algorithm 1 Finding maximum of an array of integers

---

```
1: procedure MAX( $A[0..n-1]$ )  $\triangleright$   $A[0..n-1]$  adalah array input yang terdiri dari  $n$  elemen (diindeks  
   dari 0 s.d.  $n-1$ )  
2:    $\max \leftarrow A[0]$   
3:   for  $i = 1$  to  $n-1$  do  
4:     if  $A[i] > \max$  then  
5:        $\max \leftarrow A[i]$   
6:     end if  
7:   end for  
8: end procedure
```

---

Berapakah kompleksitas algoritma di atas?  $\mathcal{O}(n)$

## 2.2. Sequential search

## 2. Sequential search (1)

**Permasalahan.** Diberikan array bilangan bulat  $(a_1, a_2, \dots, a_n)$  dan sebuah bilangan  $x$ .

Kita ingin menyelidiki apakah elemen  $x$  termuat di dalam array.

- Jika  $x$  ditemukan, algoritma menampilkan indeks posisi  $x$  dalam array.
- Jika tidak, algoritma memberikan output  $-1$ .

**Pendekatan brute-force:** bandingkan setiap elemen dalam array dengan  $x$ . Algoritma berhenti jika  $x$  ditemukan atau semua elemen telah diperiksa.

## 2. Sequential search (2)

---

**Algorithm 2** Finding an element in an array of integers

---

```
1: procedure SEQSEARCH( $A[0..n-1], x$ )
2:    $i \leftarrow 0$ 
3:   while  $i < n - 1$  and  $A[i] \neq x$  do
4:      $i \leftarrow i + 1$ 
5:   end while
6:   if  $A[i] = x$  then
7:      $idx \leftarrow i$ 
8:   else
9:      $idx \leftarrow -1$ 
10:  end if
11:  return  $idx$ 
12: end procedure
```

---

Berapakah kompleksitas algoritma di atas? Kerjakan sebagai latihan!

## 2.3. Powering (perpangkatan)

### 3. Powering (perpangkatan) (1)

**Permasalahan.** Diberikan bilangan riil  $a > 0$  dan bilangan bulat tak negatif  $n$ . Bagaimana menghitung nilai  $a^n$ ?

**Pendekatan brute-force:**

$$a^n = a \times a \times \cdots \times a \quad n \text{ times}$$

Kita mengalikan 1 dengan  $a$  sebanyak  $n$  kali.

---

**Algorithm 3** Computing  $a^n$

---

```
1: procedure POWER( $a, n$ )
2:   result  $\leftarrow$  1
3:   for  $i = 1$  to  $n$  do
4:     result  $\leftarrow$  result *  $a$ 
5:   end for
6:   return result
7: end procedure
```

---

### 3. Powering (perpangkatan) (2)

---

**Algorithm 5** Computing  $a^n$ 

---

```
1: procedure POWER( $a, n$ )
2:   result  $\leftarrow$  1
3:   for  $i = 1$  to  $n$  do
4:     result  $\leftarrow$  result *  $a$ 
5:   end for
6:   return result
7: end procedure
```

---

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

Bisakah Anda menjelaskan alasannya?

Apakah ada algoritma yang lebih baik untuk “Powering (perpangkatan)”?

## 2.4. Menghitung faktorial



## 4. Menghitung faktorial (1)

**Permasalahan.** Hitunglah nilai dari  $n!$  (dimana  $n$  adalah bilangan bulat non-negatif).

**Brute-force approach:**

$$n! = 1 \times 2 \times \cdots \times n \text{ and } 0! = 1$$

Kita mengalikan bilangan bulat 1, 2, hingga  $n$ .

## 4. Menghitung faktorial (2)

---

### Algorithm 4 Computing $n!$

---

```
1: procedure FACTORIAL( $n$ )
2:   result  $\leftarrow$  1
3:   if  $n \leq 1$  then return result
4:   else
5:     for  $i = 2$  to  $n$  do
6:       result  $\leftarrow$  result *  $i$ 
7:     end for
8:   end if
9:   return result
10: end procedure
```

---

Kompleksitas waktu:  $\mathcal{O}(n)$ . Dapatkah Anda jelaskan mengapa?

## 2.5. Perkalian matriks persegi

## 5. Perkalian matriks persegi (1)

**Permasalahan.** Diberikan dua matriks persegi, berukuran  $n \times n$ .  
Temukan cara untuk mengalikan kedua matriks tersebut dengan algoritma brute force!

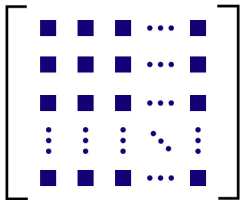
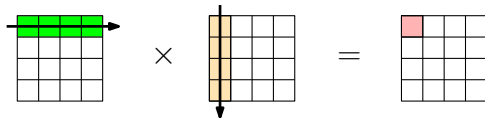


Figure: Sebuah matriks persegi

## 5. Perkalian matriks persegi (2)

Misalkan  $A = [a_{ij}]$ ,  $B = [b_{ij}]$  adalah matriks berukuran  $n \times n$ , dan  $C = A \times B$ .

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^n a_{ik}b_{kj}$$



**Pendekatan brute-force:** hitung setiap elemen  $C$  satu per satu dengan mengalikan baris yang sesuai dari  $A$  dan kolom  $B$ .

## 5. Perkalian matriks persegi (3)

---

### Algorithm 5 Perkalian matriks persegi

---

```
1: procedure MATRIXMULT( $A, B$ )
2:   for  $i \leftarrow 1$  to  $n$  do
3:     for  $j \leftarrow 1$  to  $n$  do
4:        $C[i, j] \leftarrow 0$ 
5:       for  $k \leftarrow 1$  to  $n$  do
6:          $C[i, j] \leftarrow C[i, j] + A[i, k] * B[k, j]$ 
7:       end for
8:     end for
9:   end for
10:  return  $C$ 
11: end procedure
```

---

## 5. Perkalian matriks persegi (4)

---

**Algorithm 9** Square matrix multiplication

---

```
1: procedure MATRIXMULT( $A, B$ )
2:   for  $i \leftarrow 1$  to  $n$  do
3:     for  $j \leftarrow 1$  to  $n$  do
4:        $C[i, j] \leftarrow 0$ 
5:       for  $k \leftarrow 1$  to  $n$  do
6:          $C[i, j] \leftarrow C[i, j] + A[i, k] * B[k, j]$ 
7:       end for
8:     end for
9:   end for
10:  return  $C$ 
11: end procedure
```

---

Kompleksitas waktu:  $\mathcal{O}(n^3)$

Operasi yang “dominan” (memiliki kompleksitas terbesar) dapat dihitung sebagai berikut:

- Loop terdalam memuat:  $n^3$  perkalian,  $n^3$  penjumlahan, dan 3 *assignment*
- Loop kedua memuat:  $n^2$  *assignment*
- **Return** hanya membutuhkan satu operasi

## 5. Perkalian matriks persegi (4)

Implementasikan algoritma `MATRIXMULT(A,B)` secara manual dengan mengambil matriks  $A$  berukuran  $2 \times 2$  dan matriks  $B$  berukuran  $2 \times 3$ .

Cek kebenaran implementasi Anda. Jelaskan di depan kelas.

**Jawab:**



## 2.6. Uji bilangan prima

## 6. Uji bilangan prima (1)

**Permasalahan.** Diberi bilangan bulat positif  $n$ . Periksa apakah  $n$  merupakan bilangan prima.

**Catatan.** Bilangan  $n$  adalah *prima* jika pembagi  $n$  hanyalah 1 dan  $n$ .

**1. Pendekatan brute-force:** bagi  $n$  dengan  $2, 3, \dots, n - 1$ . Jika tidak ada bilangan yang dapat membagi  $n$ , maka  $n$  adalah bilangan prima.

## 6. Uji bilangan prima (2)

---

### Algorithm 6 Uji bilangan prima

---

```
1: procedure ISPRIME( $n$ )
2:   if  $n < 2$  then
3:     return False
4:   else
5:     isprime  $\leftarrow$  True;  $k \leftarrow 2$ 
6:     while isprime & ( $k \leq n - 1$ ) do
7:       if  $n \bmod k == 0$  then
8:         isprime  $\leftarrow$  False
9:       else
10:         $k \leftarrow k + 1$ 
11:      end if
12:    end while
13:    return isprime
14:  end if
15: end procedure
```

---

## 6. Uji bilangan prima (3)

**Permasalahan.** Diberi bilangan bulat positif  $n$ . Periksa apakah  $n$  prima.

**Catatan.** Bilangan  $n$  adalah bilangan prima jika dan hanya jika pembagi dari  $n$  hanyalah 1 dan  $n$ .

- 1. Brute-force approach:** bagi  $n$  dengan  $2, 3, \dots, n - 1$ . Jika tidak ada yang membagi  $n$ , maka  $n$  adalah bilangan prima.
- 2. Sieve of Eratosthenes:** untuk batas atas tertentu  $n$ , tandai kelipatan bilangan prima secara iteratif sebagai *komposit*. Proses ini berlanjut hingga  $p \leq \sqrt{n}$ , di mana  $p$  adalah bilangan prima.

Dengan teknik ini, untuk memeriksa bahwa  $n$  adalah bilangan prima, kita hanya perlu memeriksa apakah ada bilangan prima  $\leq \sqrt{n}$  yang membagi  $n$ .

## 6. Uji bilangan prima (4)

---

### Algorithm 7 Uji bilangan prima (*sieve of Eratosthenes*)

---

```
1: procedure ISPRIME2( $n$ )
2:   if  $n < 2$  then
3:     return False
4:   else
5:     isprime  $\leftarrow$  True;  $k \leftarrow 2$ 
6:     while isprime & ( $k \leq \sqrt{n}$ ) do
7:       if  $n \bmod k == 0$  then
8:         isprime  $\leftarrow$  False
9:       else
10:         $k \leftarrow k + 1$ 
11:      end if
12:    end while
13:    return isprime
14:  end if
15: end procedure
```

---

## 6. Uji bilangan prima (5)

### Kompleksitas waktu:

**Algorithm 8** Prime number test

```
1: procedure ISPRIME( $n$ )
2:   if  $n < 2$  then
3:     return False
4:   else
5:     isprime  $\leftarrow$  True;  $k \leftarrow 2$ 
6:     while test & ( $k \leq n - 1$ ) do
7:       if  $n \bmod k == 0$  then
8:         isprime  $\leftarrow$  False
9:       else
10:         $k \leftarrow k + 1$ 
11:      end if
12:    end while
13:    return test
14:  end if
15: end procedure
```

**Algorithm 9** Prime number test (sieve of Eratosthenes)

```
1: procedure ISPRIME2( $n$ )
2:   if  $n < 2$  then
3:     return False
4:   else
5:     isprime  $\leftarrow$  True;  $k \leftarrow 2$ 
6:     while test & ( $k \leq \sqrt{n}$ ) do
7:       if  $n \bmod k == 0$  then
8:         isprime  $\leftarrow$  False
9:       else
10:         $k \leftarrow k + 1$ 
11:      end if
12:    end while
13:    return test
14:  end if
15: end procedure
```

- Brute-force:  $\mathcal{O}(n)$
- Sieve-Erathosthenes brute-force:  $\mathcal{O}(\sqrt{n})$

## 2.7. Evaluasi polinomial<sup>†</sup>

---

<sup>†</sup>Evaluasi polinomial adalah masalah penghitungan nilai polinomial pada suatu titik (nilai variabel) tertentu.

## 7. Evaluasi polinomial (1)

**Permasalahan.** Tentukan nilai polinomial berikut untuk  $x = t$ :

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

**Pendekatan brute-force:**

- 1 Untuk setiap  $i \in [0, n]$ , nilai  $t^i$  dihitung seperti pada "algoritma powering";
- 2 Kalikan  $t^k$  dengan  $a_k$ ;
- 3 Jumlahkan dengan suku lainnya.



## 7. Evaluasi polinomial (2)

---

### Algorithm 8 Evaluasi polinomial

---

```
1: procedure POLYNOM( $n, A[0..n], t$ )   ▷  $A[0..n]$  digunakan untuk menyimpan nilai konstanta
    $a_0, a_1, \dots, a_n$ 
2:    $p \leftarrow 0$ 
3:   for  $i \leftarrow n$  downto 0 do
4:     power  $\leftarrow 1$                                ▷ 'power' is used to store the value of  $t^i$  for each iteration  $i$ 
5:     for  $j \leftarrow 1$  to  $i$  do
6:       power  $\leftarrow$  power *  $t$ 
7:     end for
8:      $p \leftarrow p + A[i] * \text{power}$ 
9:   end for
10:  return  $p$ 
11: end procedure
```

---

Terdapat sebanyak  $\mathcal{O}\left(\frac{n(n+1)}{2}\right) + \mathcal{O}(n+1)$  operasi. Jadi, kompleksitas waktunya adalah:  $\mathcal{O}(n^2)$ .

## 2.8. Masalah pasangan titik terdekat

## 8. Masalah pasangan titik terdekat (1)

**Permasalahan.** Diberikan  $n$  titik pada ruang Euclid 2 dimensi (dapat dipandang sebagai sistem koordinat Cartesian). Temukan dua titik dengan jarak terdekat.

Jarak antara dua titik  $p_1 = (x_1, y_1)$  dan  $p_2 = (x_2, y_2)$  ditentukan dengan formula:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

**Pendekatan brute-force:**

- 1 Hitung jarak antara setiap pasangan titik
- 2 Ambil pasangan dengan jarak minimum

## 8. Masalah pasangan titik terdekat (2)

---

### Algorithm 9 Finding the closest points

---

```
1: procedure CLOSESTPOINTS( $p_1, p_2, \dots, p_n$ )
2:    $d_{\min} \leftarrow 999999$ 
3:   for  $i \leftarrow 1$  to  $n - 1$  do
4:     for  $j \leftarrow i + 1$  to  $n$  do
5:        $d \leftarrow \sqrt{(p_i.x - p_j.x)^2 + (p_i.y - p_j.y)^2}$ 
6:       if  $d < d_{\min}$  then
7:          $d_{\min} \leftarrow d$ 
8:          $A \leftarrow p_i$ 
9:          $B \leftarrow p_j$ 
10:      end if
11:    end for
12:  end for
13:  return  $A$  and  $B$ 
14: end procedure
```

---

## 8. Masalah pasangan titik terdekat (3)

---

**Algorithm 13** Finding the closest points

---

```
1: procedure CLOSESTPOINTS( $p_1, p_2, \dots, p_n$ )
2:    $d_{\min} \leftarrow 999999$ 
3:   for  $i \leftarrow 1$  to  $n - 1$  do
4:     for  $j \leftarrow i + 1$  to  $n$  do
5:        $d \leftarrow \sqrt{(p_i.x - p_j.x)^2 + (p_i.y - p_j.y)^2}$ 
6:       if  $d < d_{\min}$  then
7:          $d_{\min} \leftarrow d$ 
8:          $A \leftarrow p_i$ 
9:          $B \leftarrow p_j$ 
10:      end if
11:    end for
12:  end for
13:  return  $A$  and  $B$ 
14: end procedure
```

---

Kompleksitas waktu:  $O(n^2)$

## 2.9. Pencocokan pola (*pattern matching*)

## 9. Pencocokan pola (*pattern matching*) (1)

**Permasalahan.** Diberikan *sebuah string dengan panjang  $n$*  dan *sebuah pola dengan panjang  $m$*  dimana  $m < n$ . Temukan lokasi karakter pertama dari pola pada string yang cocok dengan pola tersebut.

**Contoh.**

- **String:** NOBODY NOTICED HIM
- **Pattern:** NOT

**Brute-force approach:**

- 1 Mulailah dengan karakter pertama dari string.
- 2 Mulai dari karakter pertama pada pola, periksa apakah pola tersebut cocok dengan suatu *substring*:
  - ▶ semua karakter cocok
  - ▶ ada karakter yang tidak cocok
- 3 Jika polanya tidak cocok, kita pindah ke kanan, dan ulangi Langkah 2.

## 9. Pencocokan pola (*pattern matching*) (2)

### Contoh 1.

```
N O B O D Y _ N O T I C E D _ H I M
N O T
  N O T
    N O T
      N O T
        N O T
          N O T
            N O T
              N O T
                N O T
```

### Contoh 2.

- 10010101001011110101010001
- 001011

10010101**001011**11010101010001 (*Langkah pencocokan dapat dilakukan seperti pada Contoh 1.*)



## 9. Pencocokan pola (*pattern matching*) (3)

---

```
1: procedure PATTERNMATCHING( $P, T$ )  $\triangleright$   $P$  is the array of alphabets in the pattern, and  $T$  is the  
   array of alphabets of the string  
2:    $i \leftarrow 0$ ; found  $\leftarrow$  False  $\triangleright$  'found' is a boolean variable to indicate if a the pattern has found in the  
   string, and  $i$  is the index used for the array of string's alphabets  
3:   while ( $i \leq n - m$ ) & (not found) do  
4:      $j \leftarrow 1$   $\triangleright$   $j$  is the index used for the array of pattern's alphabets  
5:     while ( $j \leq m$ ) and  $P_j = T_{i+j}$  do  
6:        $j \leftarrow j + 1$   
7:     end while  
8:     if  $j = m$  then found  $\leftarrow$  True  
9:     else  $i \leftarrow i + 1$   
10:    end if  
11:  end while  
12:  if found then return  $i + 1$   
13:  else return  $-1$   
14:  end if  
15: end procedure
```

---

## 9. Pencocokan pola (*pattern matching*) (4)

### Kompleksitas waktu

#### 1 Worst case

- ▶ Dalam setiap percobaan pencocokan, kita mencocokkan semua karakter dari pola dengan karakter dalam karakter yang sesuai dari string  $\rightarrow$  terdapat  $m$  langkah
- ▶ Ini dilakukan untuk semua kemungkinan posisi dalam string  $\rightarrow$  terdapat  $n - m + 1$  kemungkinan
- ▶ Jumlah langkah:  $m(n - m + 1) \in \mathcal{O}(nm)$

#### 2 Best case

- ▶ Ini terjadi ketika pola ditemukan di posisi  $m$  pertama dari string
- ▶ Dalam hal ini, kita memeriksa semua karakter pola
- ▶ Kompleksitas:  $\mathcal{O}(m)$

# Bagian 3: Karakteristik algoritma brute force

# Karakteristik algoritma brute force (1)

## Keunggulan algoritma brute-force:

- Ini bukan algoritma yang *powerfull*, tetapi hampir semua masalah dapat diselesaikan menggunakan algoritma brute force.
- Sederhana dan mudah dimengerti.
- Dapat diterapkan untuk banyak masalah: pencarian, pengurutan, pencocokan string, perkalian matriks, dll.
- menghasilkan algoritma standar untuk tugas komputasi seperti perkalian/penambahan angka  $n$ , menemukan maks/ment dalam larik.

## Karakteristik algoritma brute force (2)

### Kelemahan algoritma brute-force:

- Algoritmanya **tidak “smart”**, karena membutuhkan banyak perhitungan dan memakan waktu lama untuk memprosesnya. Untuk banyak masalah dunia nyata, banyaknya kandidat biasanya sangat banyak.
- Algoritma brute force **cocok untuk instance kecil**, karena sederhana dan dapat diimplementasikan dengan mudah.

**Catatan.** Algoritma ini sering disebut sebagai *algoritma naif*, dan digunakan untuk membandingkan dengan algoritma canggih lainnya.

*to be continued...*