Algoritma *Brute Force*
(Bagian 1)

Oleh: Rinaldi Munir

Bahan Kuliah
IF2251 Strategi Algoritmik

Definisi *Brute Force*

- *Brute force* : pendekatan yang lempang (*straightforward*) untuk memecahkan suatu masalah

- Biasanya didasarkan pada:
  - pernyataan masalah (*problem statement*)
  - definisi konsep yang dilibatkan.

- Algoritma *brute force* memecahkan masalah dengan
  - sangat sederhana,
  - langsung,
  - jelas (*obvious way*).
- *Just do it!*

Contoh-contoh

(Berdasarkan pernyataan masalah)

1. **Mencari elemen terbesar (terkecil)**

   Persoalan: Diberikan sebuah senarai yang beranggotakan *n* buah bilangan bulat (a_1, a_2, …, a_n). Carilah elemen terbesar di dalam senarai tersebut.

   Algoritma *brute force*: bandingkan setiap elemen senarai untuk menemukan elemen terbesar

```
procedure CariElemenTerbesar(input a1, a2, ..., an : integer,
                             output maks : integer)
{ Mencari elemen terbesar di antara elemen a1, a2, ..., an. Elemen
terbesar akan disimpan di dalam maks.
Masukan: a1, a2, ..., an
Keluaran: maks
}
Deklarasi
  k : integer

Algoritma:
  maks←a1
  for k←2 to n do
    if ak > maks then
      maks←ak
    endif
  endfor
```

Kompleksitas waktu algoritma: $O(n)$.

2. *Pencarian beruntun (Sequential Search)*

Persoalan: Diberikan senarai yang berisi *n* buah bilangan bulat (a_1, a_2, …, a_n). Carilah nilai *x* di dalam senara tersebut. Jika *x* ditemukan, maka keluarannya adalah indeks elemen senarai, jika *x* tidak ditemukan, maka keluarannya adalah 0.

Algoritma *brute force (sequential serach):* setiap elemen senarai dibandingkan dengan *x*. Pencarian selesai jika x ditemukan atau elemen senarai sudah habis diperiksa.

```
procedure PencarianBeruntun(input a1, a2, ..., an : integer,
                            x : integer,
                           output idx : integer)
{ Mencari x di dalam elemen a1, a2, ..., an. Lokasi (indeks elemen)
tempat x ditemukan diisi ke dalam idx. Jika x tidak ditemukan, maka
idx diisi dengan 0.
  Masukan: a1, a2, ..., an
  Keluaran: idx
}
Deklarasi
  k : integer

Algoritma:
  k←1
  while (k < n) and (ak ≠ x) do
     k ← k + 1
  endwhile
  { k = n or ak = x }

  if ak = x then   { x ditemukan }
     idx←k
  else
     idx← 0        { x tidak ditemukan }
  endif
```

Kompleksitas waktu algoritma: $O(n)$.

Adakah algoritma pencarian elemen yang lebih mangkus daripada *brute force*?

Contoh-contoh

(Berdasarkan definisi konsep yang terlibat)

1. **Menghitung a^n ($a > 0$, n adalah bilangan bulat tak-negatif)**

   Definisi:

   $a^n = a \times a \times \ldots \times a$ (n kali) , jika $n > 0$

   $= 1$, jika $n = 0$

   Algoritma *brute force*: kalikan 1 dengan a sebanyak n kali

```
function pangkat(a : real, n : integer) → real
{ Menghitung a^n }
Deklarasi
  i : integer
  hasil : real
Algoritma:
  hasil ← 1
  for i ← 1 to n do
     hasil ← hasil * a
  end
  return hasil
```

2. **Menghitung *n*! (*n* bilangan bulat tak-negatif)**

Definisi:

$$n! = 1 \times 2 \times 3 \times \ldots \times n \quad \text{, jika } n > 0$$
$$= 1 \quad \text{, jika } n = 0$$

Algoritma *brute force*: kalikan *n* buah bilangan, yaitu 1, 2, 3, …, *n*, bersama-sama

```
function faktorial(n : integer) → integer
{ Menghitung n! }
Deklarasi
  i : integer
  fak : real
Algoritma:
  fak ← 1
  for i ← 1 to n do
     fak ← fak * i
  end
  return fak
```

3. Mengalikan dua buah matriks, *A* dan *B*

Definisi:

Misalkan $C = A \times B$ dan elemen-elemen matrik dinyatakan sebagai c_{ij}, a_{ij}, dan b_{ij}

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

- Algoritma *brute force*: hitung setiap elemen hasil perkalian satu per satu, dengan cara mengalikan dua vektor yang panjangnya *n*.

```
procedure PerkalianMatriks(input A, B : Matriks,
                           input n : integer,
                           output C : Matriks)
{ Mengalikan matriks A dan B yang berukuran n × n, menghasilkan
  matriks C yang juga berukuran n × n
  Masukan: matriks integer A dan B, ukuran matriks n
  Keluaran: matriks C
}

Deklarasi
  i, j, k : integer

Algoritma
   for i←1 to n do
     for j←1 to n do
        C[i,j]←0    { inisialisasi penjumlah }
        for k ← 1 to n do
           C[i,j]←C[i,j] + A[i,k]*B[k,j]
        endfor
     endfor
   endfor
```

Adakah algoritma perkalian matriks yang lebih mangkus daripada *brute force*?

4. Menemukan semua faktor dari bilangan bulat *n* (selain dari 1 dan *n* itu sendiri).

- Definisi: Bilangan bulat *a* adalah faktor dari bilangan bulat *b* jika *a* habis membagi *b*.

- Algoritma *brute force*: bagi *n* dengan setiap *i* = 2, 3, …, *n* – 1. Jika *n* habis membagi *i*, maka *i* adalah faktor dari *n*.

```
procedure CariFaktor(input n : integer)
{ Mencari faktor dari bilangan bulat n selain 1 dan n itu
sendiri.
Masukan: n
Keluaran: setiap bilangan yang menjadi faktor n dicetak.
}
Deklarasi
  k : integer

Algoritma:
  k ← 1
  for k ← 2 to n - 1 do
    if n mod k = 0 then
      write(k)
    endif
  endfor
```

Adakah algoritma pemfaktoran yang lebih baik daripada *brute force*?

5. Uji keprimaan

Persoalan: Diberikan sebuah bilangan bilangan bulat positif. Ujilah apakah bilangan tersebut merupakan bilangan prima atau bukan.

> **Definisi**: bilangan prima adalah bilangan yang hanya habis dibagi oleh 1 dan dirinya sendiri.

Algoritma *brute force*: bagi n dengan 2 sampai $n - 1$. Jika semuanya tidak habis membagi n, maka n adalah bilangan prima.

Perbaikan: cukup membagi dengan 2 sampai $\sqrt{n}$ saja

```
function Prima(input x : integer)→boolean
{ Menguji apakah x bilangan prima atau bukan.
  Masukan: x
  Keluaran: true jika x prima, atau false jika x tidak prima.
}
Deklarasi
  k, y : integer
  test : boolean

Algoritma:
  if x < 2 then    { 1 bukan prima }
    return false
  else
    if x = 2 then  { 2 adalah prima, kasus khusus }
      return true
    else
      y←⌈√x⌉
      test←true
      while (test) and (y ≥ 2) do
        if x mod y = 0 then
          test←false
        else
          y←y - 1
        endif
      endwhile
      { not test or y < 2 }

      return test
   endif
 endif
```

Adakah algoritma pengujian bilangan prima yang lebih mangkus daripada *brute force*?

Algoritma Pengurutan *Brute Force*

- Algoritma apa yang paling lempang dalam memecahkan masalah pengurutan?

  Bubble sort dan *selection sort!*

- Kedua algoritma ini memperlihatkan teknik *brute force* dengan jelas sekali.

Bubble Sort

- Mulai dari elemen ke-n:
  1. Jika $s_n < s_{n-1}$, pertukarkan
  2. Jika $s_{n-1} < s_{n-2}$, pertukarkan
  …
  3. Jika $s_2 < s_1$, pertukarkan

→ 1 kali *pass*

- Ulangi lagi untuk pass ke-*i*, tetapi sampai elemen ke-*i*
- Semuanya ada *n* – 1 kali *pass*

```
procedure BubbleSort (input/output s : TabelInt, input n : integer)
{ Mengurutkan tabel s[1..N] sehingga terurut menaik dengan metode
  pengurutan bubble sort.

  Masukan : Tabel s yang sudah terdefenisi nilai-nilainya.
  Keluaran: Tabel s yang terurut menaik sedemikian sehingga
                s[1] £ s[2] £ … £ s[N].
}
Deklarasi
   i    : integer     { pencacah untuk jumlah langkah }
   k    : integer     { pencacah,untuk pengapungan pada setiap
 langkah }
   temp : integer     { peubah bantu untuk pertukaran }

Algoritma:
   for i ← 1 to n - 1 do
     for k ← n downto i + 1 do
        if s[k] < s[k-1] then
          {pertukarkan s[k] dengan s[k-1]}
           temp ← s[k]
           s[k] ← s[k-1]
           s[k-1] ← temp
         endif
     endfor
   endfor
```

Kompleksitas waktu algoritma: $O(n^2)$.
Adakah algoritma pengurutan elemen elemen yang lebih mangkus?

Selection Sort

Pass ke –1:

1. Cari elemen terbesar mulai di dalam *s*[1..*n*]
2. Letakkan elemen terbesar pada posisi n (pertukaran)

P*ass* ke-2:

1. Cari elemen terbesar mulai di dalam *s*[1..*n* - 1]
2. Letakkan elemen terbesar pada posisi *n* - 1 (pertukaran)

Ulangi sampai hanya tersisa 1 elemen

Semuanya ada *n* –1 kali *pass*

```
procedure PengurutanSeleksi(input/output  s : array [1..n] of integer)
{ Mengurutkan s1, s2, ..., sn sehingga tersusun menaik dengan metode pengurutan seleksi.
  Masukan: s1, s2, ..., sn
  Keluaran: s1, s2, ..., sn (terurut menaik)
}
Deklarasi
    i, j, imaks, temp : integer

Algoritma:
    for i ← n downto 2 do   { jumlah pass sebanyak n – 1 }
       { cari elemen terbesar di dalam s[1], s[2], ..., s[i] }
       imaks ← 1    { elemen pertama diasumsikan sebagai elemen  terbesar sementara
       for j ← 2 to i do
         if s[j] > s[imaks] then
            imaks ← j
         endif
       endfor
       {pertukarkan s[imaks] dengan s[i] }
       temp ← s[i]
       s[i] ← s[imaks]
       s[imaks] ← temp
    endfor
```

Kompleksitas waktu algoritma: $O(n^2)$.
Adakah algoritma pengurutan elemen elemen yang lebih mangkus?

Mengevaluasi polinom

Persoalan: Hitung nilai polinom

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

untuk $x = t$.

Algoritma *brute force*: x^i dihitung secara *brute force* (seperti perhitungan a^n). Kalikan nilai x^i dengan a_i, lalu jumlahkan dengan suku-suku lainnya.

```
function polinom(input t : real)→real
{ Menghitung nilai p(x) pada x = t. Koefisien-koefisein polinom
sudah disimpan di dalam a[0..n].
Masukan: t
Keluaran: nilai polinom pada x = t.
}
Deklarasi
  i, j : integer
  p, pangkat  : real

Algoritma:
  p ← 0
  for i ← n downto 0 do
     pangkat ← 1
     for j ← 1 to i do  {hitung x^i }
        pangkat ← pangkat * t
     endfor
     p ← p + a[i] * pangkat
  endfor
  return p
```

Kompleksitas algoritma ini adalah $O(n^2)$.

Perbaikan (*improve*):

```
function polinom2(input x0 : real)→real
{ Menghitung nilai p(x) pada x = t. Koefisien-koefisein polinom
sudah disimpan di dalam a[0..n].
Masukan: x0
Keluaran: nilai polinom pada x = t.
}
Deklarasi
  i, j : integer
  p, pangkat  : real

Algoritma:
  p ← a[n]
  pangkat←1
  for i ← 1 to n do
     pangkat ← pangkat * t
     p ← p + a[i] * pangkat
  endfor
  return p
```

Kompleksitas algoritma ini adalah $O(n)$.
Adakah algoritma perhitungan nilai polinom yang lebih mangkus daripada *brute force*?

Karakteristik Algoritma *Brute Force*

1. Algoritma *brute force* umumnya tidak “cerdas” dan tidak mangkus, karena ia membutuhkan jumlah langkah yang besar dalam penyelesaiannya.

   Kata “force” mengindikasikan “tenaga” ketimbang “otak”

   Kadang-kadang algoritma *brute force* disebut juga algoritma naif (*naïve algorithm*).

3. Algoritma *brute force* lebih cocok untuk masalah yang berukuran kecil.

Pertimbangannya:

- sederhana,
- implementasinya mudah

Algoritma *brute force* sering digunakan sebagai basis pembanding dengan algoritma yang lebih mangkus.

4. Meskipun bukan metode yang mangkus, hampir semua masalah dapat diselesaikan dengan algoritma *brute force*.

   Sukar menunjukkan masalah yang tidak dapat diselesaikan dengan metode *brute force*.

   Bahkan, ada masalah yang hanya dapat diselesaikan dengan metode *brute force*.

   Contoh: mencari elemen terbesar di dalam senarai.

   Contoh lainnya?