Algoritma *Brute Force*

Definisi *Brute Force*

- *Brute force* adalah sebuah pendekatan yang lempang (*straightforward*) untuk memecahkan suatu masalah, biasanya didasarkan pada pernyataan masalah (*problem statement*) dan definisi konsep yang dilibatkan.

- Algoritma *brute force* memecahkan masalah dengan sangat sederhana, langsung dan dengan cara yang jelas (*obvious way*).

Contoh-contoh *Brute Force*

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

$$a^n = a \times a \times \ldots \times a \quad (n \text{ kali}) \text{, jika } n > 0$$
$$= 1 \qquad \text{, jika } n = 0$$

Algoritma: kalikan 1 dengan a sebanyak n kali

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

$$n! = \begin{cases} 1 \times 2 \times 3 \times \ldots \times n & \text{, jika } n > 0 \\ 1 & \text{, jika } n = 0 \end{cases}$$

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

3. Mengalikan dua buah matrik yang berukuran $n \times n$.

- 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: 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*.

```
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
  ketemu ← false
  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. **Mencari elemen terbesar (atau terkecil)**

Persoalan: Diberikan sebuah himpunan yang beranggotakan n buah bilangan bulat. Bilangan-bilangan bulat tersebut dinyatakan sebagai $a_1, a_2, \ldots, a_n$. Carilah elemen terbesar di dalam himpunan tersebut.

```
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 algoritma ini adalah $O(n)$.

6. ***Sequential Search***

Persoalan: Diberikan *n* buah bilangan bulat yang dinyatakan sebagai a_1, a_2, …, a_n. Carilah apakah x terdapat di dalam himpunan bilangan bulat tersebut. Jika *x* ditemukan, maka lokasi (indeks) elemen yang bernilai *x* disimpan di dalam peubah *idx*. Jika *x* tidak terdapat di dalam himpunan tersebut, maka *idx* diisi dengan nilai 0.

```
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 algoritma ini adalah $O(n)$.

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

7. *Bubble Sort*

- Apa metode yang paling lempang dalam memecahkan masalah pengurutan? Jawabnya adalah algoritma pengurutan *bubble sort*.

- Algoritma *bubble sort* mengimplementasikan teknik *brute force* dengan jelas sekali.

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

   Masukan : Tabel L yang sudah terdefenisi nilai-nilainya.
   Keluaran: Tabel L yang terurut menaik sedemikian sehingga
                  L[1] ≤ L[2] ≤ … ≤ L[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 L[k] < L[k-1] then
           {pertukarkan L[k] dengan L[k-1]}
            temp ←  L[k]
            L[k] ←  L[k-1]
            L[k-1] ← temp
          endif
      endfor
    endfor
```

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

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

8. Uji keprimaan

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

```
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*?

9. Menghitung nilai polinom secara *brute force*

Persoalan: Hitung nilai polinom

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

pada titik $x = x_0$.

```
function polinom(input x0 : real)→real
{ Menghitung nilai p(x) pada x = x0. Koefisien-koefisein polinom sudah
disimpan di dalam tabel a. Derajat polinom (n) juga sudah terdefinisi.
Masukan: x0
Keluaran: nilai polinom pada x = x0.
}
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 * x0
      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 = x0. Koefisien-koefisein polinom sudah
disimpan di
  dalam tabel a. Derajat polinom (n) juga sudah terdefinisi.
  Masukan: x0
  Keluaran: nilai polinom pada x = x0.
}
Deklarasi
  i, j : integer
  p, pangkat  : real

Algoritma:
  p←a0
  pangkat←1
  for i←1 to n do
     pangkat←pangkat * x0
     p←p + ai * 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. Kadang-kadang algoritma *brute force* disebut juga algoritma naif (*naïve algorithm*).

2. Algoritma *brute force* seringkali merupakan pilihan yang kurang disukai karena ketidakmangkusannya itu, tetapi dengan mencari pola-pola yang mendasar, keteraturan, atau trik-trik khusus, biasanya akan membantu kita menemukan algoritma yang lebih cerdas dan lebih mangkus.

3. Untuk masalah yang ukurannya kecil, kesederhanaan *brute force* biasanya lebih diperhitungkan daripada ketidakmangkusannya.

   Algoritma *brute force* sering digunakan sebagai basis bila membandingkan beberapa alternatif algoritma yang mangkus.

4. Algoritma *brute force* seringkali lebih mudah diimplementasikan daripada algoritma yang lebih canggih, dan karena kesederhanaannya, kadang-kadang algoritma *brute force* dapat lebih mangkus (ditinjau dari segi implementasi).