

# Algoritma *Greedy*

## Pendahuluan

- Algoritma *greedy* merupakan metode yang paling populer untuk memecahkan persoalan optimasi.
- Persoalan optimasi (*optimization problems*):  
→ persoalan mencari solusi optimum.
- Hanya ada dua macam persoalan optimasi:
  1. Maksimasi (*maximization*)
  2. Minimasi (*minimization*)

Contoh persoalan optimasi:

( **Masalah Penukaran Uang**): Diberikan uang senilai  $A$ . Tukar  $A$  dengan koin-koin uang yang ada. Berapa jumlah minimum koin yang diperlukan untuk penukaran tersebut?

→ Persoalan minimasi

**Contoh 1:** tersedia banyak koin 1, 5, 10, 25

- Uang senilai  $A = 32$  dapat ditukar dengan banyak cara berikut:
  - $32 = 1 + 1 + \dots + 1$  (32 koin)
  - $32 = 5 + 5 + 5 + 5 + 10 + 1 + 1$  (7 koin)
  - $32 = 10 + 10 + 10 + 1 + 1$  (5 koin)
  - ... dst
- Minimum:  $32 = 25 + 5 + 1 + 1$  (4 koin)

- *Greedy* = rakus, tamak, loba, ...
- Prinsip *greedy*: “*take what you can get now!*”.
- Algoritma *greedy* membentuk solusi langkah per langkah (*step by step*).
- Pada setiap langkah, terdapat banyak pilihan yang perlu dieksplorasi.
- Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan.

- Pada setiap langkah, kita membuat pilihan **optimum lokal** (*local optimum*)
- dengan harapan bahwa langkah sisanya mengarah ke solusi **optimum global** (*global optimum*).

- Algoritma *greedy* adalah algoritma yang memecahkan masalah langkah per langkah;

pada setiap langkah:

1. mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan (prinsip "*take what you can get now!*")
2. berharap bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global.

- Tinjau masalah penukaran uang:

Strategi *greedy*:

Pada setiap langkah, pilihlah koin dengan nilai terbesar dari himpunan koin yang tersisa.

- Misal:  $A = 32$ , koin yang tersedia: 1, 5, 10, dan 25  
*Langkah 1*: pilih 1 buah koin 25 (Total = 25)  
*Langkah 2*: pilih 1 buah koin 5 (Total =  $25 + 5 = 30$ )  
*Langkah 3*: pilih 2 buah koin 1 (Total =  $25 + 5 + 1 + 1 = 32$ )
- Solusi: Jumlah koin minimum = 4 (solusi optimal!)

Elemen-elemen algoritma greedy:

1. Himpunan kandidat,  $C$ .
2. Himpunan solusi,  $S$
3. Fungsi seleksi (*selection function*)
4. Fungsi kelayakan (*feasible*)
5. Fungsi obyektif

Dengan kata lain:

algoritma *greedy* melibatkan pencarian sebuah himpunan bagian,  $S$ , dari himpunan kandidat,  $C$ ; yang dalam hal ini,  $S$  harus memenuhi beberapa kriteria yang ditentukan, yaitu menyatakan suatu solusi dan  $S$  dioptimisasi oleh fungsi obyektif.

Pada masalah penukaran uang:

- *Himpunan kandidat*: himpunan koin yang merepresentasikan nilai 1, 5, 10, 25, paling sedikit mengandung satu koin untuk setiap nilai.
- *Himpunan solusi*: total nilai koin yang dipilih tepat sama jumlahnya dengan nilai uang yang ditukarkan.
- *Fungsi seleksi*: pilihlah koin yang bernilai tertinggi dari himpunan kandidat yang tersisa.
- *Fungsi layak*: memeriksa apakah nilai total dari himpunan koin yang dipilih tidak melebihi jumlah uang yang harus dibayar.
- *Fungsi obyektif*: jumlah koin yang digunakan minimum.

## Skema umum algoritma *greedy*:

```

function greedy(input C: himpunan_kandidat) → himpunan_kandidat
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy
  Masukan: himpunan kandidat C
  Keluaran: himpunan solusi yang bertipe himpunan_kandidat
}
Deklarasi
  x : kandidat
  S : himpunan_kandidat

Algoritma:
  S ← {} { inisialisasi S dengan kosong }
  while (not SOLUSI(S)) and (C ≠ {} ) do
    x ← SELEKSI(C) { pilih sebuah kandidat dari C }
    C ← C - {x} { elemen himpunan kandidat berkurang satu }
    if LAYAK(S ∪ {x}) then
      S ← S ∪ {x}
    endif
  endwhile
  { SOLUSI(S) or C = {} }

  if SOLUSI(S) then
    return S
  else
    write('tidak ada solusi')
  endif

```

- Pada akhir setiap lelaran, solusi yang terbentuk adalah optimum lokal.
- Pada akhir kalang while-do diperoleh optimum global.

- *Warning*: Optimum global belum tentu merupakan solusi optimum (terbaik), tetapi *sub-optimum* atau *pseudo-optimum*.
- Alasan:
  1. Algoritma *greedy* tidak beroperasi secara menyeluruh terhadap semua alternatif solusi yang ada (sebagaimana pada metode *exhaustive search*).
  2. Terdapat beberapa fungsi SELEKSI yang berbeda, sehingga kita harus memilih fungsi yang tepat jika kita ingin algoritma menghasilkan solusi optimal.
- Jadi, pada sebagian masalah algoritma *greedy* tidak selalu berhasil memberikan solusi yang optimal.

- **Contoh 2:** tinjau masalah penukaran uang.

- (a) Koin: 5, 4, 3, dan 1  
 Uang yang ditukar = 7.  
 Solusi *greedy*:  $7 = 5 + 1 + 1$  (3 koin) → tidak optimal  
 Solusi optimal:  $7 = 4 + 3$  (2 koin)
- (b) Koin: 10, 7, 1  
 Uang yang ditukar: 15  
 Solusi *greedy*:  $15 = 10 + 1 + 1 + 1 + 1 + 1$  (6 koin)  
 Solusi optimal:  $15 = 7 + 7 + 1$  (hanya 3 koin)
- (c) Koin: 15, 10, dan 1  
 Uang yang ditukar: 20  
 Solusi *greedy*:  $20 = 15 + 1 + 1 + 1 + 1 + 1$  (6 koin)  
 Solusi optimal:  $20 = 10 + 10$  (2 koin)

- Untuk sistem mata uang dollar AS, euro Eropa, dan *crown* Swedia, algoritma *greedy* selalu memberikan solusi optimum.
- Contoh: Uang \$6,39 ditukar dengan uang kertas (*bill*) dan koin sen (*cent*), kita dapat memilih:
  - Satu buah uang kertas senilai \$5
  - Satu buah uang kertas senilai \$1
  - Satu koin 25 sen
  - Satu koin 10 sen
  - Empat koin 1 sen

$$\$5 + \$1 + 25c + 10c + 1c + 1c + 1c + 1c = \$6,39$$

- Jika jawaban terbaik mutlak tidak diperlukan, maka algoritma *greedy* sering berguna untuk menghasilkan solusi hampiran (*approximation*), daripada menggunakan algoritma yang lebih rumit untuk menghasilkan solusi yang eksak.
- Bila algoritma *greedy* optimum, maka keoptimalannya itu dapat dibuktikan secara matematis

## Contoh-contoh Algoritma Greedy

### 1. Masalah penukaran uang

Nilai uang yang ditukar:  $A$

Himpunan koin (*multiset*):  $\{d_1, d_2, \dots, d_n\}$ .

Himpunan solusi:  $X = \{x_1, x_2, \dots, x_n\}$ ,

$x_i = 1$  jika  $d_i$  dipilih,  $x_i = 0$  jika  $d_i$  tidak dipilih.

Obyektif persoalan adalah

$$\text{Minimisasi } F = \sum_{i=1}^n x_i \quad (\text{fungsi obyektif})$$

$$\text{dengan kendala } \sum_{i=1}^n d_i x_i = A$$



## Penyelesaian dengan *exhaustive search*

- Terdapat  $2^n$  kemungkinan solusi (nilai-nilai  $X = \{x_1, x_2, \dots, x_n\}$  )
- Untuk mengevaluasi fungsi obyektif =  $O(n)$
- Kompleksitas algoritma *exhaustive search* seluruhnya =  $O(n \cdot 2^n)$ .

## Penyelesaian dengan algoritma *greedy*

- Strategi *greedy*: Pada setiap langkah, pilih koin dengan nilai terbesar dari himpunan koin yang tersisa.

```
function CoinExchange(input C : himpunan_koin, A : integer) → himpunan_koin
{ mengembalikan koin-koin yang total nilainya = A, tetapi jumlah koinnya minimum }
```

### Deklarasi

```
S : himpunan_koin
x : koin
```

### Algoritma

```
S ← {}
while (Σ(nilai semua koin di dalam S) ≠ A) and (C ≠ {} ) do
  x ← koin yang mempunyai nilai terbesar
  C ← C - {x}
  if (Σ(nilai semua koin di dalam S) + nilai koin x ≤ A then
    S ← S ∪ {x}
  endif
endwhile

if (Σ(nilai semua koin di dalam S) = A then
  return S
else
  write('tidak ada solusi')
endif
```

- Agar pemilihan koin berikutnya optimal, maka perlu mengurutkan himpunan koin dalam urutan yang menurun (*nonincreasing order*).
- Jika himpunan koin sudah terurut menurun, maka kompleksitas algoritma *greedy* =  $O(n)$ .
- Sayangnya, algoritma *greedy* untuk masalah penukaran uang ini tidak selalu menghasilkan solusi optimal (lihat contoh sebelumnya).

## 2. Minimisasi Waktu di dalam Sistem (Penjadwalan)

- Persoalan: Sebuah *server* (dapat berupa *processor*, pompa, kasir di bank, dll) mempunyai  $n$  pelanggan (*customer*, *client*) yang harus dilayani. Waktu pelayanan untuk setiap pelanggan  $i$  adalah  $t_i$ .

Minimumkan total waktu di dalam sistem:

$$T = \sum_{i=1}^n (\text{waktu di dalam sistem})$$

- Ekuivalen dengan meminimumkan waktu rata-rata pelanggan di dalam sistem.

**Contoh 3:** Tiga pelanggan dengan

$$t_1 = 5, \quad t_2 = 10, \quad t_3 = 3,$$

Enam urutan pelayanan yang mungkin:

Urutan	$T$
1, 2, 3:	$5 + (5 + 10) + (5 + 10 + 3) = 38$
1, 3, 2:	$5 + (5 + 3) + (5 + 3 + 10) = 31$
2, 1, 3:	$10 + (10 + 5) + (10 + 5 + 3) = 43$
2, 3, 1:	$10 + (10 + 3) + (10 + 3 + 5) = 41$
<b>3, 1, 2:</b>	<b><math>3 + (3 + 5) + (3 + 5 + 10) = 29 \leftarrow \text{(optimal)}</math></b>
3, 2, 1:	$3 + (3 + 10) + (3 + 10 + 5) = 34$

### Penyelesaian dengan *Exhaustive Search*

- Urutan pelanggan yang dilayani oleh server merupakan suatu permutasi
- Jika ada  $n$  orang pelanggan, maka terdapat  $n!$  urutan pelanggan
- Untuk mengevaluasi fungsi obyektif :  $O(n)$
- Kompleksitas algoritma *exhaustive search* =  $O(nn!)$

## Penyelesaian dengan algoritma *greedy*

- Strategi *greedy*: Pada setiap langkah, pilih pelanggan yang membutuhkan waktu pelayanan terkecil di antara pelanggan lain yang belum dilayani.

```
function PenjadwalanPelanggan(input C : himpunan_pelanggan) → himpunan_pelanggan
{ mengembalikan urutan jadwal pelayanan pelanggan yang meminimumkan waktu di dalam sistem }
```

### Deklarasi

```
S : himpunan_pelanggan
i : pelanggan
```

### Algoritma

```
S ← {}
while (C ≠ {}) do
  i ← pelanggan yang mempunyai t[i] terkecil
  C ← C - {i}
  S ← S ∪ {i}
endwhile
return S
```

- Agar proses pemilihan pelanggan berikutnya optimal, urutkan pelanggan berdasarkan waktu pelayanan dalam urutan yang menaik.
- Jika pelanggan sudah terurut, kompleksitas algoritma *greedy* =  $O(n)$ .

```
procedure PenjadwalanPelanggan(input n:integer)
{ Mencetak informasi deretan pelanggan yang akan diproses oleh server tunggal
  Masukan: n pelanggan, setiap pelanggan dinomori 1, 2, ..., n
  Keluaran: urutan pelanggan yang dilayani
}
Deklarasi
  i : integer
Algoritma:
  {pelanggan 1, 2, ..., n sudah diurut menaik berdasarkan ti}
  for i←1 to n do
    write('Pelanggan ', i, ' dilayani!')
  endfor
```

- Algoritma *greedy* untuk penjadwalan pelanggan akan selalu menghasilkan solusi optimum.
- **Teorema.** Jika  $t_1 \leq t_2 \leq \dots \leq t_n$  maka pengurutan  $i_j = j$ ,  $1 \leq j \leq n$  meminimumkan

$$T = \sum_{k=1}^n \sum_{j=1}^k t_{i_j}$$

untuk semua kemungkinan permutasi  $i_j$ .

### 3. Integer Knapsack

$$\text{Maksimasi } F = \sum_{i=1}^n p_i x_i$$

dengan kendala (*constraint*)

$$\sum_{i=1}^n w_i x_i \leq K$$

yang dalam hal ini,  $x_i = 0$  atau  $1$ ,  $i = 1, 2, \dots, n$

## Penyelesaian dengan *exhaustive search*

- Sudah dijelaskan pada pembahasan *exhaustive search*.
- Kompleksitas algoritma *exhaustive search* untuk persoalan ini =  $O(n \cdot 2^n)$ .

## Penyelesaian dengan algoritma *greedy*

- Masukkan objek satu per satu ke dalam *knapsack*. Sekali objek dimasukkan ke dalam *knapsack*, objek tersebut tidak bisa dikeluarkan lagi.
- Terdapat beberapa strategi *greedy* yang heuristik yang dapat digunakan untuk memilih objek yang akan dimasukkan ke dalam *knapsack*:

1. *Greedy by profit.*

- Pada setiap langkah, pilih objek yang mempunyai keuntungan terbesar.
- Mencoba memaksimumkan keuntungan dengan memilih objek yang paling menguntungkan terlebih dahulu.

2. *Greedy by weight.*

- Pada setiap langkah, pilih objek yang mempunyai berat teringan.
- Mencoba memaksimumkan keuntungan dengan memasukkan sebanyak mungkin objek ke dalam *knapsack*.

3. *Greedy by density.*

- Pada setiap langkah, *knapsack* diisi dengan objek yang mempunyai  $p_i/w_i$  terbesar.
  - Mencoba memaksimumkan keuntungan dengan memilih objek yang mempunyai keuntungan per unit berat terbesar.
- Pemilihan objek berdasarkan salah satu dari ketiga strategi di atas tidak menjamin akan memberikan solusi optimal.

**Contoh 4.**

$w_1 = 2; p_1 = 12; w_2 = 5; p_2 = 15;$

$w_3 = 10; p_3 = 50; w_4 = 5; p_4 = 10$

Kapasitas *knapsack*  $K = 16$

Properti objek				Greedy by			Solusi Optimal
$i$	$w_i$	$p_i$	$p_i/w_i$	$profit$	$weight$	$density$	
1	6	12	2	0	1	0	0
2	5	15	3	1	1	1	1
3	10	50	5	1	0	1	1
4	5	10	2	0	1	0	0
Total bobot				15	16	15	15
Total keuntungan				65	37	65	65

- Solusi optimal:  $X = (0, 1, 1, 0)$
- Greedy by profit dan greedy by density memberikan solusi optimal!

**Contoh 5.**

$w_1 = 100; p_1 = 40; w_2 = 50; p_2 = 35; w_3 = 45; p_3 = 18;$

$w_4 = 20; p_4 = 4; w_5 = 10; p_5 = 10; w_6 = 5; p_6 = 2$

Kapasitas *knapsack*  $K = 100$

Properti objek				Greedy by			Solusi Optimal
$i$	$w_i$	$p_i$	$p_i/w_i$	$profit$	$weight$	$density$	
1	100	40	0,4	1	0	0	0
2	50	35	0,7	0	0	1	1
3	45	18	0,4	0	1	0	1
4	20	4	0,2	0	1	1	0
5	10	10	1,0	0	1	1	0
6	5	2	0,4	0	1	1	0
Total bobot				100	80	85	100
Total keuntungan				40	34	51	55

Ketiga strategi gagal memberikan solusi optimal!



**Kesimpulan:** Algoritma *greedy* tidak selalu berhasil menemukan solusi optimal untuk masalah 0/1 *Knapsack*.

#### 4. Fractional Knapsack

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

dengan kendala (*constraint*)

$$\sum_{i=1}^n w_i x_i \leq K$$

yang dalam hal ini,  $0 \leq x_i \leq 1, i = 1, 2, \dots, n$

### Penyelesaian dengan *exhaustive search*

- Oleh karena  $0 \leq x_i \leq 1$ , maka terdapat tidak berhingga nilai-nilai  $x_i$ .
- Persoalan *Fractional Knapsack* menjadi malar (*continuous*) sehingga tidak mungkin dipecahkan dengan algoritma *exhaustive search*.

### Penyelesaian dengan algoritma *greedy*

- Ketiga strategi *greedy* yang telah disebutkan di atas dapat digunakan untuk memilih objek yang akan dimasukkan ke dalam *knapsack*.

**Contoh 6.**

$w_1 = 18; \quad p_1 = 25; \quad w_2 = 15; \quad p_2 = 24$   
 $w_3 = 10; \quad p_3 = 15 \quad \text{Kapasitas knapsack } K = 20$

Properti objek				Greedy by		
$i$	$w_i$	$p_i$	$p_i/w_i$	$profit$	$weight$	$density$
1	18	25	1,4	1	0	0
2	15	24	1,6	2/15	2/3	1
3	10	15	1,5	0	1	1/2
Total bobot				20	20	20
Total keuntungan				28,2	31,0	31,5

- Solusi optimal:  $X = (0, 1, 1/2)$
- yang memberikan keuntungan maksimum = 31,5.

- Strategi pemilihan objek berdasarkan densitas  $p_i/w_i$  terbesar akan selalu memberikan solusi optimal.
- Agar proses pemilihan objek berikutnya optimal, maka kita urutkan objek berdasarkan  $p_i/w_i$  yang menurun, sehingga objek berikutnya yang dipilih adalah objek sesuai dalam urutan itu.

**Teorema 3.2.** Jika  $p_1/w_1 \geq p_2/w_2 \geq \dots \geq p_n/w_n$  maka algoritma *greedy* dengan strategi pemilihan objek berdasarkan  $p_i/w_i$  terbesar menghasilkan solusi yang optimum.

- Algoritma persoalan *fractional knapsack*:

1. Hitung harga  $p_i/w_i$ ,  $i = 1, 2, \dots, n$
2. Urutkan seluruh objek berdasarkan nilai  $p_i/w_i$  dari besar ke kecil
3. Panggil `FractinonalKnapsack`

```
function FractionalKnapsack(input C : himpunan_objek, K : real) → himpunan_solusi
/ Menghasilkan solusi persoalan fractional knapsack dengan algoritma greedy yang
menggunakan strategi pemilihan objek berdasarkan density (pi/wi). Solusi dinyatakan
sebagai vektor X = x[1], x[2], ..., x[n].

Asumsi: Seluruh objek sudah terurut berdasarkan nilai pi/wi yang menurun
)

Deklarasi
  i, TotalBobot : integer
  MasihMuatUtuh : boolean
  x : himpunan_solusi

Algoritma:
  for i ← 1 to n do
    x[i] ← 0 { inisialisasi setiap fraksi objek i dengan 0 }
  endfor

  i ← 0
  TotalBobot ← 0
  MasihMuatUtuh ← true
  while (i ≤ n) and (MasihMuatUtuh) do
    { tinjau objek ke-i }
    i ← i + 1
    if TotalBobot + C.w[i] ≤ K then
      { masukkan objek i ke dalam knapsack }
      x[i] ← 1
      TotalBobot ← TotalBobot + C.w[i]
    else
      MasihMuatUtuh ← false
      x[i] ← (K - TotalBobot)/C.w[i]
    endif
  endwhile
  { i > n or not MasihMuatUtuh }

  return x
```

Kompleksitas waktu algoritma =  $O(n)$ .