# 8 - Teknik Greedy (part 1)

[KOMS120403]

Desain dan Analisis Algoritma (2022/2023)

Dewi Sintiari

Prodi S1 Ilmu Komputer
Universitas Pendidikan Ganesha

Week 9 (April 2023)

# Daftar isi

# **Bagian 1.** Masalah optimisasi

# Masalah optimisasi

Masalah optimisasi adalah masalah menemukan solusi terbaik dari semua solusi yang "layak" (*feasible*).

**Tipe masalah optimisasi**

- Maksimisasi
  - Contoh: Integer knapsack problem
- Minimisasi
  - Contoh: Masalah pewarnaan graf (*graph coloring*), TSP

**Bentuk standar**:

$$\begin{aligned} \text{minimalkan} \quad & f(x) \\ \text{atas kendala} \quad & g_i(x) \leq 0, \ i = 1, \ldots, m \\ & h_j(x) = 0, \ j = 1, \ldots, p \end{aligned}$$

# Algoritma Greedy

## Definition

Algoritma Greedy adalah paradigma algoritma yang membangun solusi **sepotong demi sepotong**, dengan cara memilih bagian berikutnya yang **menawarkan manfaat paling jelas dan langsung** sesuai dengan fungsi tujuannya.

Ini adalah algoritma yang paling populer dan paling sederhana untuk memecahkan masalah optimisasi (yakni maksimisasi dan minimisasi).

- Prinsip algoritma *greedy*: **Ambil apa yang bisa kamu dapatkan sekarang!**
- Algoritma Greedy membuat solusi selangkah demi selangkah.
- Di setiap langkah, ada banyak kemungkinan pilihan. Kita mengambil keputusan terbaik di setiap langkah, yaitu kita memilih optimum lokal, untuk mencapai optimum global.

# Algoritma Greedy

## Contoh (Masalah penukaran koin)

*Diberikan cek sebesar $42. Kita ingin menukarkan cek tersebut dengan uang koin. Berapa jumlah minimum koin yang dibutuhkan dalam pertukaran, jika nominal koin yang tersedia adalah 1, 5, 10, dan 25.*

Ada banyak kemungkinan kombinasi koin. Dengan algoritma brute-force, kita dapat dengan mudah membuat daftar semua kemungkinan:

- $42 = 1 + 1 + \cdots + 1 \rightarrow 42$ koin
- $42 = 5 + 1 + 1 + \cdots + 1 \rightarrow 38$ koin
- ... dst.

Dengan algoritma greedy, pada setiap langkah, kita mengambil koin dengan nilai maksimum sebanyak mungkin.

$$42 = 25 + 10 + 5 + 1 + 1$$

Jadi, lima koin cukup untuk menukar cek tersebut.

# **Bagian 2.** Skema algoritma Greedy

# Komponen algoritma Greedy

- Himpunan kandidat - terdiri dari calon solusi yang dibuat dari himpunan.
- Fungsi seleksi - digunakan untuk memilih kandidat terbaik untuk ditambahkan ke solusi.
- Fungsi fesibilitas - digunakan untuk menentukan apakah kandidat dapat dimasukkan dalam solusi (layak/tidak layak).
- Fungsi objektif - digunakan untuk menetapkan nilai ke solusi atau solusi parsial (memaksimalkan atau meminimalkan).
- Fungsi solusi - digunakan untuk menunjukkan apakah solusi lengkap telah tercapai.

# Skema algoritma Greedy

**Analisis komponen masalah penukaran koin:**

- Himpunan kandidat (*candidate set*): himpunan koin $\{1, 5, 10, 25\}$.
- Fungsi seleksi (*selection function*): pilih koin yang memiliki nilai maksimum.
- Fungsi kelayakan (*feasibility function*): periksa apakah jumlah koin setelah mengambil koin baru tidak melebihi jumlah uang.
- Fungsi objektif (*objective function*): meminimalkan jumlah koin yang digunakan.
- Fungsi solusi (*solution function*): himpunan koin yang dipilih $\{1, 10, 25\}$.

# Skema algoritma Greedy

**Algorithm 1** General Skema algoritma Greedy

```
1:  procedure GREEDY(C: candidate set)
2:      S ← {}                                   ▷ S is the solution function
3:      while (not SOLUTION(S)) and (C ≠ {}) do
4:          x ← SELECTION(C)
5:          C ← C − {x}
6:          if FEASIBLE(S ∪ {x}) then
7:              S ← S ∪ {x}
8:          end if
9:      end while
10:     if SOLUTION(S) then return S
11:     else print('No solution exists')
12:     end if
13: end procedure
```

- Pada akhir iterasi (“ kondisi if”), kita memiliki solusi optimum lokal.
- Di akhir while loop, kita mendapatkan solusi optimum global (jika ada).

# Penyelesaian masalah dengan algoritma greedy

**Beberapa contoh:**

1. Masalah penukaran koin
2. Masalah pemilihan aktivitas
3. Minimisasi waktu dalam sistem
4. Integer knapsack problem
5. Fractional knapsack problem
6. Huffman coding
7. Traveling Salesman Problem

# Contoh 1.

## Masalah penukaran koin

# 1. Masalah penukaran koin (1)

**Formulasi masalah:**

- Jumlah uang yang ingin ditukarkan: $M$
- Kumpulan koin yang tersedia: $\{c_1, c_2, \ldots, c_n\}$.
- Kumpulan solusi: $X = \{x_1, x_2, \ldots, x_n\}$, di mana $x_i = 1$ jika $a_i$ dipilih dan $x_i = 0$ jika tidak.

Fungsi objektif:

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

$$\text{dengan kendala } \sum_{i=1}^{n} c_i x_i = M$$

# 1. Masalah penukaran koin (2)

**Penyelesaian dengann exhaustive search**

- Karena $X = \{x_1, x_2, \ldots, x_n\}$ dan $x_i \in \{01\}$, maka ada $2^n$ kemungkinan solusi.
- Untuk mengevaluasi fungsi tujuan untuk setiap kandidat solusi, kita memerlukan $\mathcal{O}(n)$-time.
- Jadi, kompleksitas waktu dari pencarian menyeluruh adalah: $\mathcal{O}(n \cdot 2^n)$.

# 1. Masalah penukaran koin (3)

**Algorithm 2** Skema umum algoritma Greedy

```
1:  procedure CoinExchange(C: coin set, M: integer)
2:      S ← {}
3:      while (∑(all coins in S) ≠ M) and (C ≠ {}) do
4:          x ← coin of maximum value
5:          C ← C − {x}
6:          if ∑(all coins in S) + value(x) ≤ M then
7:              S ← S ∪ {x}
8:          end if
9:      end while
10:     if ∑(all coins in S) = M then
11:         return S
12:     else
13:         print('No feasible solution')
14:     end if
15: end procedure
```

# 1. Masalah penukaran koin (4)

**Kompleksitas waktu:**

- Memilih koin dengan nilai maksimum: $\mathcal{O}(n)$ (menggunakan kekerasan untuk mendapatkan maks).
- Perulangan while diulang $n$ kali (maksimum), jadi kerumitan keseluruhannya adalah: $\mathcal{O}(n^2)$.
- Jika koin diurutkan dalam urutan menurun, kerumitannya menjadi $\mathcal{O}(n)$, karena memilih koin dengan nilai maksimal hanya membutuhkan $\mathcal{O}(1)$-waktu.

# Contoh 2.

## Masalah pemilihan aktivitas

# 2. Masalah pemilihan aktivitas (1)

**Masalah:** diberikan $n$ aktivitas $S = \{1, 2, \ldots, n\}$, yang akan menggunakan sumber daya (misalnya, ruang rapat, studio, prosesor, dll.).

Misalkan sumber daya hanya dapat digunakan untuk melakukan satu aktivitas pada satu waktu. Setiap kali suatu aktivitas menempati sumber daya, aktivitas lain tidak dapat menggunakannya hingga aktivitas pertama selesai.

Setiap aktivitas $i$ dimulai pada waktu $s_i$ dan berakhir pada waktu $f_i$, di mana $s_i \leq f_i$. Dua aktivitas $i$ dan $j$ disebut *kompatibel* jika interval $[s_i, f_i]$ dan $[s_j, f_j]$ tidak berpotongan.

Tujuan kita adalah memilih sebanyak mungkin aktivitas yang dapat dilayani oleh sumber daya.

# 2. Masalah pemilihan aktivitas (2)

## Contoh (Masalah pemilihan aktivitas)

*Diberikan $n = 11$ aktivitas dengan waktu mulai-berakhir seperti yang diberikan pada tabel berikut:*

| $i$ | $s_i$ | $f_i$ |
|---|---|---|
| 1 | 1 | 4 |
| 2 | 3 | 5 |
| 3 | 4 | 6 |
| 4 | 5 | 7 |
| 5 | 3 | 8 |
| 6 | 7 | 9 |
| 7 | 10 | 11 |
| 8 | 8 | 12 |
| 9 | 8 | 13 |
| 10 | 2 | 14 |
| 11 | 13 | 15 |

# 2. Masalah pemilihan aktivitas (3)

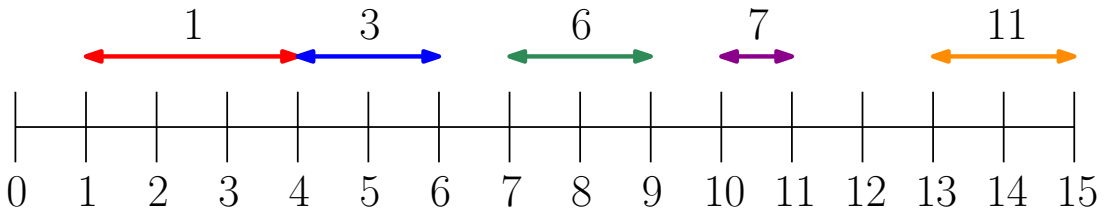
**Pendekatan dengan exhaustive search:**

1. Buat daftar semua himpunan bagian dari himpunan aktivitas $n$.
2. Evaluasi setiap subset, periksa apakah solusinya kompatibel.
3. Jika ya, maka subset adalah kandidat solusi.
4. Pilih kandidat solusi dengan jumlah aktivitas maksimum.
5. Kompleksitas waktu dari algoritma adalah $\mathcal{O}(n \cdot 2^n)$. Mengapa?

# 2. Masalah pemilihan aktivitas (4)

**Pendekatan dengan algoritma Greedy:**

1. Urutkan aktivitas berdasarkan waktu akhir dalam urutan menaik;
2. Pada setiap langkah, pilih aktivitas yang waktu mulainya lebih besar atau sama dengan waktu akhir aktivitas yang dipilih sebelumnya.




**Strategi:** pada setiap langkah, kita memilih aktivitas indeks terkecil yang dapat dilakukan dalam slot waktu yang tersedia.

Himpunan solusi $\{1, 3, 6, 7, 11\}$

# 2. Masalah pemilihan aktivitas (5)

**Algorithm 3** Greedy Activity Selector

```
1:  procedure ActvtySlctr((s_1, ..., s_n), (f_1, ..., f_n))
2:      n ← length(s)                          ▷ s = (s_1, ..., s_n)
3:      A ← {1}                                ▷ A is the solution function
4:      j ← 1
5:      for i ← 2 to n do
6:          if s_i ≥ f_j then                  ▷ s_i: starting time of activity i, f(j): finishing time of activity j
7:              A ← A ∪ {i}
8:              j ← i
9:          end if
10:     end for
11: end procedure
```

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

# 2. Masalah pemilihan aktivitas (6)

**Teorema**

*Algoritma greedy memberikan solusi optimal untuk "Masalah pemilihan aktivitas".*

**Ide pembuktian.**

- Kita ingin menunjukkan bahwa jadwal $A$ yang dipilih oleh greedy adalah optimal.
- Untuk melakukan ini, kita menunjukkan bahwa jumlah aktivitas di $A$ setidaknya sebesar jumlah aktivitas di rangkaian aktivitas lain yang tidak tumpang tindih.
- Untuk menunjukkan ini, misalkan $B$ adalah rangkaian aktivitas sebarang dan tidak saling tumpang tindih. Kita menunjukkan bahwa kita dapat mengganti setiap aktivitas di $B$ dengan aktivitas di $A$.

# 2. Masalah pemilihan aktivitas (7)

## Proof.

Misalkan $S = \{1, 2, \ldots, n\}$ adalah rangkaian aktivitas yang diurutkan berdasarkan waktu penyelesaian, $A$ adalah solusi optimal, dan $B$ adalah keluaran dari algoritma greedy. $A$ dan $B$ diurutkan berdasarkan waktu selesainya

Misalkan $a_x$ adalah aktivitas pertama di $A$ yang berbeda dari aktivitas di $B$. Jadi:

- $A = a_1, a_2, \ldots, a_{x-1}, a_x, a_{x+1}, \ldots$
- $B = a_1, a_2, \ldots, a_{x-1}, b_x, b_{x+1}, \ldots$

Karena $B$ dipilih oleh algoritma Greedy, $b_x$ memiliki waktu selesai lebih awal dari waktu selesai $a_x$.

Jadi, $A' = A - \{a_x\} \cup \{b_x\} = a_1, a_2, \ldots, a_{x-1}, b_x, b_{x+1}, \ldots$ (yaitu mengganti $a_x$ dengan $b_x$ di $A$) juga merupakan solusi yang layak.

Dengan melanjutkan proses ini, kita dapat melihat bahwa kita dapat mengganti masing-masing aktivitas di $A$ dengan aktivitas di $B$. □

# 2. Masalah pemilihan aktivitas (8)

## Alternatif pembuktian lain

**Proof.**

Misalkan $S = \{1, 2, \ldots, n\}$ adalah kumpulan aktivitas yang diurutkan berdasarkan waktu penyelesaian.

Misalkan $A \subseteq S$ adalah solusi optimal, dimana elemen di $A$ juga diurutkan berdasarkan waktu penyelesaian, dan elemen pertama dari $A$ adalah $k_1$.

Jika $k_1 = 1$, maka $A$ dimulai dengan pilihan greedy (seperti dalam algoritma). Jika tidak, kita menunjukkan bahwa $B = (A - \{k_1\}) \cup \{1\}$ (solusi yang dimulai dengan pilihan greedy 1 adalah solusi optimal lainnya).

Karena $f_1 \leq f_{k_1}$ (karena 1 adalah elemen pertama dari $S$), dan aktivitas di $A$ kompatibel, maka aktivitas di $B$ juga kompatibel. Karena $|A| = |B|$, maka $B$ adalah solusi optimal. Oleh karena itu, ada solusi optimal yang dimulai dengan pilihan greedy. □

# Contoh 3.

## Minimisasi waktu dalam sistem

# 3. Minimisasi waktu dalam sistem (1)

**Permasalahan:** server (prosesor, kasir, layanan pelanggan, dll.) memiliki $n$ klien yang harus dilayani. Waktu untuk melayani klien $i$ adalah $t_i$. Bagaimana meminimalkan total waktu dalam sistem (termasuk waktu tunggu)?

$$T = \sum_{i=1}^{n} \text{waktu yang dihabiskan dalam sistem}$$

*Catatan.* Masalah ini setara dengan meminimalkan waktu rata-rata klien dalam sistem.

# 3. Minimisasi waktu dalam sistem (2)

**Contoh**

*Terdapat tiga klien dengan waktu melayani:* $t_1 = 5$, $t_2 = 10$, *dan* $t_3 = 3$.

Kemungkinan urutan klien:

- **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:** $10 + (10 + 3) + (10 + 3 + 5) = 41$
- **3,1,2:** $3 + (3 + 5) + (3 + 5 + 10) = 29$
- **3,2,1:** $3 + (3 + 10) + (3 + 10 + 5) = 34$

# 3. Minimisasi waktu dalam sistem (3)

- Masalah lain yang mirip dengan meminimalkan waktu dalam sistem adalah optimal storage on tapes atau penyimpanan musik dalam kaset (sistem analog dengan sistem penyimpanan berurutan).
- Program/musik disimpan dalam kaset secara berurutan. Panjang setiap lagu $i$ adalah $t_i$ (dalam detik/menit). Untuk mengambil dan memutar lagu, kaset itu awalnya ditempatkan di awal.
- Jika lagu dalam kaset disimpan dengan urutan $X = \{x_1, x_2, \ldots, x_n\}$, maka waktu yang diperlukan untuk memutar lagu $x_j$ sampai selesai adalah $T_j = \sum_{1 \le k \le j} t_{x_k}$ .
- Jika semua lagu sering diputar, *mean retrieval time / MRT* adalah $\frac{1}{n} \sum_{1 \le j \le n} T_j$.
- Dalam hal ini kita diminta mencari permutasi dari $n$ lagu sedemikian sehingga jika lagu disimpan dalam kaset, maka MRT minimum. Meminimalkan MRT sama dengan meminimalkan hal-hal berikut:

$$d(X) = \frac{1}{n} \sum_{1 \le j \le n} \sum_{1 \le k \le j} t_{x_k} \quad \text{atau} \quad d(X) = \sum_{1 \le j \le n} \sum_{1 \le k \le j} t_{x_k}$$

# 3. Minimisasi waktu dalam sistem (4)

**Pencarian solusi dengan exhaustive search**

- Urutan klien dalam sistem adalah permutasi. Banyaknya permutasi elemen $n$ adalah $n!$.
- Untuk mengevaluasi fungsi tujuan permutasi membutuhkan $\mathcal{O}(n)$-waktu.
- Kompleksitas waktu exhaustive search adalah $\mathcal{O}(n \cdot n!)$.

# 3. Minimisasi waktu dalam sistem (5)

**Solusi untuk algoritma greedy**

**Strategi:** Di setiap langkah, pilih klien yang membutuhkan waktu pelayanan minimum di antara semua klien yang belum dilayani.

**Algorithm 4** Clients Scheduling

```
1: procedure ClientsScheduling(n)
2:     S ← {}                          ▷ S is the solution function
3:     while C ≠ {} do                 ▷ C is the solution candidate
4:         i ← client with minimum t[i] in C   ▷ t_i is the serving time of client i
5:         C ← C − {i}
6:         S ← S ∪ {i}
7:     end while
8:     return S
9: end procedure
```

Kompleksitas waktu: $\mathcal{O}(n^2)$. Coba Anda jelaskan mengapa!

# 3. Minimisasi waktu dalam sistem (6)

Jika klien diurutkan berdasarkan waktu pelayanan (dalam urutan menaik), maka kompleksitas algoritma greedy adalah $\mathcal{O}(n)$.

**Algorithm 5** Clients Scheduling

```
1: procedure ClientsScheduling2(n)
2:     input: klien (1, 2, ..., n) dipesan naik berdasarkan t_i
3:     for i ← 1 to n do
4:         print(i)
5:     end for
6: end procedure
```

*Catatan.* Algoritma greedy untuk penjadwalan klien berdasarkan waktu pelayanan dengan urutan menaik selalu menghasilkan solusi yang optimal.

# 3. Minimisasi waktu dalam sistem (7)

**Teorema**

*Jika* $t_1 \leq t_2 \leq \cdots \leq t_n$, *maka urutan* $i_j = j$, $1 \leq j \leq n$ *meminimalkan nilai*

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

*untuk setiap kemungkinan permutasi* $i_j$, $1 \leq j \leq n$.

Proof available in Ellis Horrowitz & Sartaj Sahni, Computer Algorithms, 1998). See next slide.

# 3. Minimisasi waktu dalam sistem (8)

**Proof:** Let $I = i_1, i_2, \ldots, i_n$ be any permutation of the index set $\{1, 2, \ldots, n\}$. Then

$$d(I) = \sum_{k=1}^{n} \sum_{j=1}^{k} t_{i_j} = \sum_{k=1}^{n} (n - k + 1) t_{i_k}$$

If there exist $a$ and $b$ such that $a < b$ and $t_{i_a} > t_{i_b}$, then interchanging $i_a$ and $i_b$ results in a permutation $I'$ with

$$d(I') = \left[ \sum_{\substack{k \\ k \neq a \\ k \neq b}} (n - k + 1) t_{i_k} \right] + (n - a + 1) t_{i_b} + (n - b + 1) t_{i_a}$$

Subtracting $d(I')$ from $d(I)$, we obtain

$$\begin{aligned} d(I) - d(I') &= (n - a + 1)(t_{i_a} - t_{i_b}) + (n - b + 1)(t_{i_b} - t_{i_a}) \\ &= (b - a)(t_{i_a} - t_{i_b}) \\ &> 0 \end{aligned}$$

Hence, no permutation that is not in nondecreasing order of the $t_i$'s can have minimum $d$. It is easy to see that all permutations in nondecreasing order of the $t_i$'s have the same $d$ value. Hence, the ordering defined by $i_j = j, 1 \leq j \leq n$, minimizes the $d$ value. □

*to be continued...*