6.4 - Transform and Conquer

[KOMS124404]

Desain dan Analisis Algoritma (2024/2025)

Dewi Sintiari

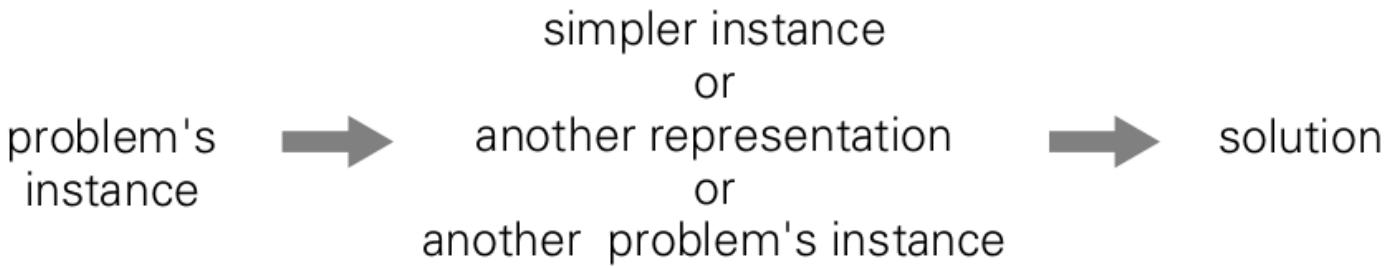
Prodi S1 Ilmu Komputer
Universitas Pendidikan Ganesha

Week 6 (Maret 2025)

Daftar isi

- Prinsip dasar Transform-and-Conquer
- Variasi Transform-and-Conquer
  - *Instance simplification* (penyederhanaan instance)
  - *Representation change* (perubahan representasi)
  - *Problem reduction* (reduksi permasalahan)

Prinsip dasar Transform-and-Conquer (1)



Prinsip dasar Transform-and-Conquer (1)

Tiga variasi pendekatan:

1. Instance simplification: transformasi ke contoh yang lebih sederhana atau lebih nyaman dari masalah yang sama.
2. Representation change: transformasi ke representasi yang berbeda dari contoh yang sama.
3. Problem reduction: transformasi ke turunan dari masalah berbeda yang algoritmanya sudah tersedia.

Bagian 1. Penyederhanaan *instance*

1. Presorting (1): pengurutan data sebelum algoritma

- Banyak permasalahan yang melibatkan *list* atau *array* yang lebih mudah dijawab jika list/array sudah terurut.
- Efisiensi waktu algoritma yang membutuhkan *sorting* (pengurutan data) seringkali bergantung pada efisiensi algoritma pengurutan yang digunakan.

Ingat kembali algoritma pengurutan data yang sudah dibahas:

- Selection sort, Bubble sort, dan Insertion sort: kompleksitas $\Theta(n^2)$.
- Merge sort dan Quick sort: kompleksitas waktu $\Theta(n \log n)$ pada kasus rata-rata tetapi kuadratik pada kasus terburuk.

1. Presorting (2): contoh

Contoh (Memeriksa keunikan elemen dalam array)

Permasalahan: *Diberikan sebuah array, periksa apakah ada elemen yang sama.*

- *Pendekatan brute-force:*

  Bandingkan pasangan elemen array *hingga dua elemen yang sama ditemukan atau tidak ada lagi pasangan yang tersisa. Kompleksitas:* $\Theta(n^2)$.

- *Dengan presorting:*
  Urutkan array terlebih dahulu lalu **periksa hanya elemen yang berurutan**: *jika array memuat elemen yang sama, maka kedua elemen pasti bersebelahan (sebab array sudah diurutkan).*

1. Presorting (2): pseudocode

Memeriksa keunikan dalam array dengan presorting

Algorithm 1 Checking uniqueness in an array

```
1:  procedure UNIQUE(A[0..n − 1])
2:      input: an ordorable array A[0..n − 1]
3:      output: "True" if A has no equal elements, "False" otherwise
4:      Sort the array A                          ▷ Implement the presorting
5:      for i ← 0 to n − 2 do
6:          if A[i] = A[i + 1] then               ▷ Compare the adjacent elements
7:              return false
8:          end if
9:      end for
10:     return true
11: end procedure
```

1. Presorting (2): *time complexity* (TC)

TC = time to sort + time to check the consecutive elements

- Waktu untuk memeriksa elemen berurutan: $\leq n - 1$

Jadi, kompleksitas waktu tergantung pada algoritma sorting yang digunakan dalam presorting

- Jika menggunakan Selection sort, dsb., maka kompleksitas waktunya $\Theta(n^2)$
- Jika menggunakan Merge sort/Quick sort, maka kompleksitas waktunya $\Theta(n \log n)$

$$T(n) = T_{sort}(n) + T_{scan}(n) \in \Theta(n \log n) + \Theta(n) = \Theta(n \log n)$$

Ini memberikan kompleksitas yang lebih baik daripada dengan pendekatan brute-force.

2. Searching problem (masalah pencarian)

- **Without sorting**: Pencarian berurutan (brute-force), kompleksitas $\Theta(n)$
- **With sorting**: binary search (rekursif dan membutuhkan array yang terurut)

$$T(n) = T_{sort}(n) + T_{search}(n) = \Theta(n \log n) + \Theta(\log n) = \Theta(n \log n)$$

Jadi dalam hal ini, Sequential Search adalah pilihan yang lebih baik.

Bagian 2. Representation change

2.1. Eliminasi Gauss

1. Eliminasi Gauss (1)

Sistem persamaan linier (yang sudah Anda pelajari):

$$\begin{cases} a_{11}x + a_{12}y &= b_1 \\ a_{21}x + a_{22}y &= b_2 \end{cases}$$

di mana: a_{ij} dan b_i untuk $i, j \in \{1, 2\}$ adalah bilangan real; x dan y adalah variabel yang tidak diketahui.

Permasalahan

Diberikan SPL dengan n persamaan dalam n variabel yang tidak diketahui:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \quad\vdots & \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{cases}$$

Tugasnya adalah menyelesaikan sistem, yaitu temukan nilainya $[x_1, x_2, \ldots, x_n]$

1. Eliminasi Gauss (2)

Algoritma penyelesaian: gunakan substitusi/eliminasi berulang seperti dalam kasus $n = 2$.

Langkah yang lebih elegan adalah dengan Eliminasi Gauss

- Idenya adalah untuk mengubah sistem persamaan linear n dalam n yang tidak diketahui menjadi sistem yang ekuivalen (yaitu, sistem dengan solusi yang sama seperti yang asli) dengan matriks koefisien segitiga atas, matriks dengan semua nol di bawah matriks utamanya. diagonal (*Anda sudah belajar di perkuliahan Aljabar Linier*).

Kita bertujuan untuk mentransfernya ke dalam bentuk berikut:

$$\begin{cases} a'_{11}x_1 + a'_{12}x_2 + \cdots + a'_{1n}x_n &= b'_1 \\ \qquad\qquad a'_{22}x_2 + \cdots + a'_{2n}x_n &= b'_2 \\ \vdots \\ \qquad\qquad\qquad\qquad a'_{nn}x_n &= b'_n \end{cases}$$

1. Eliminasi Gauss (3)

Dalam notasi matriks, ini dapat ditulis sebagai:

$$A(x) = b \Rightarrow A'x = b'$$

dimana:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}, \; b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}, \; A' = \begin{pmatrix} a'_{11} & a'_{12} & \dots & a'_{1n} \\ 0 & a'_{22} & \dots & a'_{2n} \\ \vdots & & & \\ 0 & 0 & \dots & a'_{nn} \end{pmatrix}, \; b' = \begin{pmatrix} b'_1 \\ b'_2 \\ \vdots \\ b'_n \end{pmatrix}$$

Mengapa sistem dengan matriks koefisien segitiga atas lebih baik daripada sistem dengan matriks koefisien sebarang?

1. Eliminasi Gauss (3)

Dalam notasi matriks, ini dapat ditulis sebagai:

$$A(x) = b \Rightarrow A'x = b'$$

dimana:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}, \; b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}, \; A' = \begin{pmatrix} a'_{11} & a'_{12} & \dots & a'_{1n} \\ 0 & a'_{22} & \dots & a'_{2n} \\ \vdots & & & \\ 0 & 0 & \dots & a'_{nn} \end{pmatrix}, \; b' = \begin{pmatrix} b'_1 \\ b'_2 \\ \vdots \\ b'_n \end{pmatrix}$$

Mengapa sistem dengan matriks koefisien segitiga atas lebih baik daripada sistem dengan matriks koefisien sebarang?

- Karena kita dapat dengan mudah menyelesaikan sistem dengan matriks koefisien segitiga atas dengan substitusi balik.

1. Eliminasi Gauss (4)

Penyelesaian Sistem Persamaan Linier (seperti yang dibahas di kuliah AlLin)

$$A'x = b \Leftrightarrow \begin{pmatrix} a'_{11} & a'_{12} & \dots & a'_{1(n-1)} & a'_{1n} \\ 0 & a'_{22} & \dots & a'_{2(n-1)} & a'_{1n} \\ \vdots & & & & \\ 0 & 0 & \dots & a'_{(n-1)(n-1)} & a'_{(n-1)n} \\ 0 & 0 & \dots & 0 & a'_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b'_1 \\ b'_2 \\ \vdots \\ b'_{n-1} \\ b'_n \end{pmatrix}$$

- Pertama, kita bisa langsung mencari nilai x_n dari persamaan terakhir, yaitu: $a'_{nn}x_n = b'_n$.
- Kemudian kita dapat mensubstitusi nilai ini ke persamaan terakhir, yaitu $a'_{(n-1)(n-1)}x_{n-1} + a'_{(n-1)n}x_n = b'_n$ untuk mendapatkan x_{n-1}.
- ...dan seterusnya, hingga kita mengganti nilai yang diketahui dari variabel $n-1$ terakhir ke dalam persamaan pertama, dari mana kita menemukan nilai x_1.

1. Eliminasi Gauss (5)

Bagaimana kita bisa beralih dari sistem dengan matriks koefisien arbitrer A ke sistem ekuivalen dengan matriks koefisien segitiga atas A?

Kita dapat melakukannya melalui serangkaian operasi baris elementer.

- menukar dua persamaan sistem;
- mengganti persamaan dengan kelipatan bukan nolnya;
- mengganti persamaan dengan jumlah atau selisih dari persamaan ini dan beberapa kelipatan dari persamaan lain

1. Eliminasi Gauss (6)

Penerapan Implementasi Eliminasi Gauss (A. Levitin, page 236)

EXAMPLE 1 Solve the system by Gaussian elimination.

$$
\begin{aligned}
2x_1 - x_2 + x_3 &= 1\\
4x_1 + x_2 - x_3 &= 5\\
x_1 + x_2 + x_3 &= 0.
\end{aligned}
$$

$$
\begin{bmatrix} 2 & -1 & 1 & 1 \\ 4 & 1 & -1 & 5 \\ 1 & 1 & 1 & 0 \end{bmatrix}
\begin{matrix} \\ \text{row } 2 - \frac{4}{2} \text{ row } 1 \\ \text{row } 3 - \frac{1}{2} \text{ row } 1 \end{matrix}
$$

$$
\begin{bmatrix} 2 & -1 & 1 & 1 \\ 0 & 3 & -3 & 3 \\ 0 & \frac{3}{2} & \frac{1}{2} & -\frac{1}{2} \end{bmatrix}
\quad \text{row } 3 - \frac{1}{2} \text{ row } 2
$$

$$
\begin{bmatrix} 2 & -1 & 1 & 1 \\ 0 & 3 & -3 & 3 \\ 0 & 0 & 2 & -2 \end{bmatrix}
$$

Now we can obtain the solution by back substitutions:

$$x_3 = (-2)/2 = -1, \quad x_2 = (3 - (-3)x_3)/3 = 0, \quad \text{and} \quad x_1 = (1 - x_3 - (-1)x_2)/2 = 1.$$

1. Eliminasi Gauss (7)

Untuk pseudocode dari algoritma dan menghitung kompleksitas waktu, lihat slide berikutnya (ini tidak dibahas di kelas).

Bacalah **page 236-238** pada buku A. Levitin untuk penjelasan yang lebih lengkap!

1. Eliminasi Gauss (8)

Berikut adalah pseudocode dari tahap pertama algoritma Eliminasi Gauss, yang disebut eliminasi maju.

ALGORITHM *ForwardElimination*$(A[1..n, 1..n], b[1..n])$

//Applies Gaussian elimination to matrix A of a system's coefficients,
//augmented with vector b of the system's right-hand side values
//Input: Matrix $A[1..n, 1..n]$ and column-vector $b[1..n]$
//Output: An equivalent upper-triangular matrix in place of A with the
//corresponding right-hand side values in the $(n + 1)$st column

```
for i ← 1 to n do A[i, n + 1] ← b[i]    //augments the matrix
for i ← 1 to n − 1 do
    for j ← i + 1 to n do
        for k ← i to n + 1 do
            A[j, k] ← A[j, k] − A[i, k] ∗ A[j, i] / A[i, i]
```

1. Eliminasi Gauss (9)

Perbaikan dari algoritma sebelumnya (page 237).

ALGORITHM *BetterForwardElimination*$(A[1..n, 1..n], b[1..n])$

//Implements Gaussian elimination with partial pivoting
//Input: Matrix $A[1..n, 1..n]$ and column-vector $b[1..n]$
//Output: An equivalent upper-triangular matrix in place of A and the
//corresponding right-hand side values in place of the $(n+1)$st column
for $i \leftarrow 1$ **to** n **do** $A[i, n+1] \leftarrow b[i]$ //appends b to A as the last column
for $i \leftarrow 1$ **to** $n-1$ **do**
    $pivotrow \leftarrow i$
    for $j \leftarrow i+1$ **to** n **do**
        if $|A[j, i]| > |A[pivotrow, i]|$ $pivotrow \leftarrow j$
    for $k \leftarrow i$ **to** $n+1$ **do**
        $swap(A[i, k], A[pivotrow, k])$
    for $j \leftarrow i+1$ **to** n **do**
        $temp \leftarrow A[j, i] / A[i, i]$
        for $k \leftarrow i$ **to** $n+1$ **do**
            $A[j, k] \leftarrow A[j, k] - A[i, k] * temp$

1. Eliminasi Gauss (10)

Kompleksitas waktu "*BetterForwardElimination*" (page 238).

$$
\begin{aligned}
C(n) &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{k=i}^{n+1} 1 = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (n+1-i+1) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (n+2-i) \\
&= \sum_{i=1}^{n-1} (n+2-i)(n-(i+1)+1) = \sum_{i=1}^{n-1} (n+2-i)(n-i) \\
&= (n+1)(n-1) + n(n-2) + \cdots + 3 \cdot 1 \\
&= \sum_{j=1}^{n-1} (j+2)j = \sum_{j=1}^{n-1} j^2 + \sum_{j=1}^{n-1} 2j = \frac{(n-1)n(2n-1)}{6} + 2\frac{(n-1)n}{2} \\
&= \frac{n(n-1)(2n+5)}{6} \approx \frac{1}{3}n^3 \in \Theta(n^3).
\end{aligned}
$$

2.2. Eliminasi Gauss untuk Dekomposisi LU

2. Eliminasi Gauss untuk Dekomposisi LU (1)

Contoh lain "Instance simplification"

- Lower-Upper (LU) Decomposition: memfaktorkan suatu matriks sebagai perkalian matriks segitiga bawah dan matriks segitiga atas. Contoh:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

Bagaimana dekomposisi ini berguna?

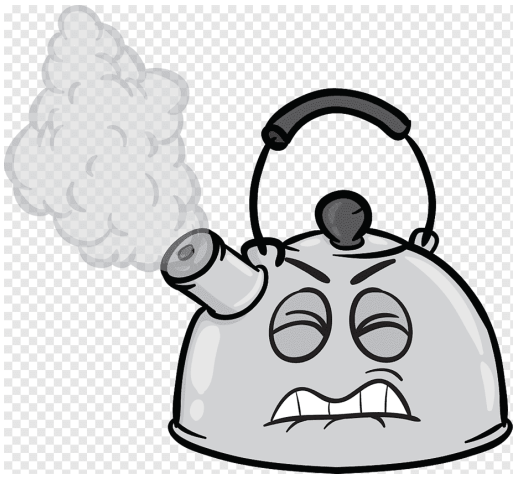
- Untuk menyelesaikan sistem linier $Ax = b$, kita dapat menguraikan A menjadi $A = LU$, jadi $Ax = b \Leftrightarrow LUx = b$
- Sekarang asumsikan $Ux = y$, jadi $LUx = b \Leftrightarrow Ly = b$
- Selesaikan sistem linier $Ly = b$, sehingga kita memperoleh y.
- Kemudian selesaikan sistem linier $Ux = y$, sehingga kita memperoleh x.
- Di sini, menyelesaikan $Ly = b$ atau $Ux = y$ dapat dilakukan dengan mudah dengan substitusi, karena L adalah matriks segitiga bawah dan U adalah matriks segitiga atas.

Bagian 3 Problem reduction

Problem reduction

A mathematician joke of problem reduction

Profesor X, seorang ahli matematika terkenal, memperhatikan bahwa ketika istrinya ingin merebus air untuk teh mereka, dia mengambil ketel dari lemari, mengisinya dengan air, dan meletakkannya di atas kompor. Suatu ketika, saat istrinya pergi, sang profesor harus merebus air sendiri. Dia melihat ketel itu ada di meja dapur. Apa yang dilakukan Profesor X? Dia meletakkan ketel di lemari terlebih dahulu dan kemudian mengikuti rutinitas istrinya.



Problem reduction

Problem reduction: Jika Anda perlu memecahkan masalah, kurangi menjadi masalah lain yang Anda ketahui cara menyelesaikannya. (*Coba jelaskan dengan ilustrasi kasus pada cerita Profesor X di atas.*)

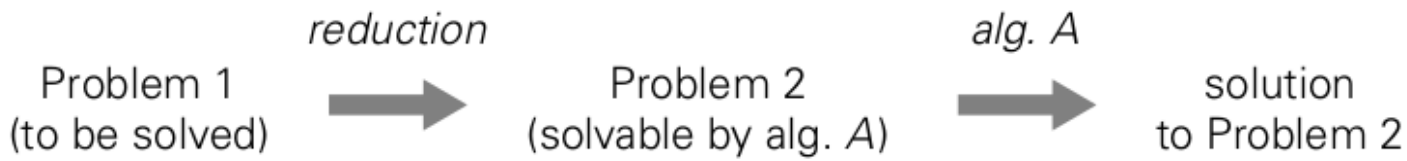


Figure: Strategi "problem reduction"

Contoh *problem reduction*

Selanjutnya, akan dibahas beberapa contoh dari penerapan *problem reduction*, yaitu:

1. Menghitung KPK
2. Menghitung banyak lintasan
3. Reduksi masalah optimasi
4. Pemrograman linier

1. Menghitung KPK (1)

Kelipatan Persekutuan Terkecil (KPK) dari dua bilangan bulat positif m dan n, dilambangkan kpk(m, n), didefinisikan sebagai bilangan bulat terkecil yang habis dibagi m dan n.

Contoh: kpk$(24, 60) = 120$, dan kpk$(11, 5) = 55$.

Metode penghitungan KPK (saat di sekolah dulu...):

- Temukan faktorisasi prima dari m dan n;
- Hitung perkalian semua faktor prima persekutuan dari m dan n;
- Kalikan dengan semua faktor prima dari m yang tidak ada di n, dan semua faktor prima dari n yang tidak ada di m.

Contoh:

$$24 = 2 \cdot 2 \cdot 2 \cdot 3$$
$$60 = 2 \cdot 2 \cdot 3 \cdot 5$$
$$\text{kpk}(24, 60) = (2 \cdot 2 \cdot 3) \cdot 2 \cdot 5 = 120$$

1. Menghitung KPK (2)

Perhatikan bahwa fpb(m, n) dapat dihitung dengan mengambil hasil perkalian dari *semua pembagi prima umum dari m dan n*.

Contoh:

$$\begin{aligned} 24 &= 2 \cdot 2 \cdot 2 \cdot 3 \\ 60 &= 2 \cdot 2 \cdot 3 \cdot 5 \\ \text{fpb}(24, 60) &= 2 \cdot 2 \cdot 3 = 12 \end{aligned}$$

Kita dapat melihat bahwa hubungan berikut berlaku:

$$m \cdot n = \text{fpb}(m, n) \cdot \text{kpk}(m, n) \Leftrightarrow \text{kpk}(m, n) = \frac{m \cdot n}{\text{fpb}(m, n)}$$

Jadi, **masalah menemukan kpk direduksi menjadi menemukan fpb**, dan ini dapat diselesaikan dengan menggunakan algoritma Euclid untuk fpb.

2. Banyaknya lintasan pada graf

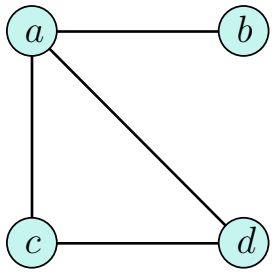
Masalah: Diberikan sebuah graf G, dan simpul v_i dan v_j di G. Temukan banyaknya lintasan di G dari v_i ke v_j.

Lemma (Banyaknya litasan (*path*))

Misalkan G adalah graf tak berarah dengan matriks ketetanggaan A, dengan urutan simpul: $v_1, v_2, \ldots, v_n$ dari $V(G)$. Banyaknya lintasan berbeda dengan panjang $r > 0$ dari v_i ke v_j, sama dengan entri (i, j)-th dari A^r.

Jadi, masalah ini direduksi menjadi *menghitung perpangkatan matriks persegi*.

Contoh:



$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & \mathrm{c} & \mathrm{d} \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array} \qquad A^2 = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & \mathrm{c} & \mathrm{d} \end{array} \\ \begin{bmatrix} 3 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix} \end{array}$$

Contoh: banyaknya lintasan dengan panjang 2 dari a ke c sama dengan entri (a, c) dari matriks A^2, yaitu 1 (satu-satunya jalur adalah $a - d - c$).

3. Reduksi Masalah Optimasi (1)

- Masalah maksimasi: masalah pencarian maksimum beberapa fungsi.
- Masalah minimisasi: masalah pencarian nilai minimum beberapa fungsi.

Misalkan sekarang Anda perlu menemukan *nilai minimum dari fungsi* $f(x)$ dan Anda memiliki algoritma untuk memaksimalkan fungsi. Bagaimana Anda bisa menggunakan algoritma tersebut untuk menyelesaikan masalah minimisasi?

We have relation

$$\min f(x) = -\max[-f(x)]$$

$$\max f(x) = -\min[-f(x)]$$

3. Reduksi Masalah Optimasi (2)

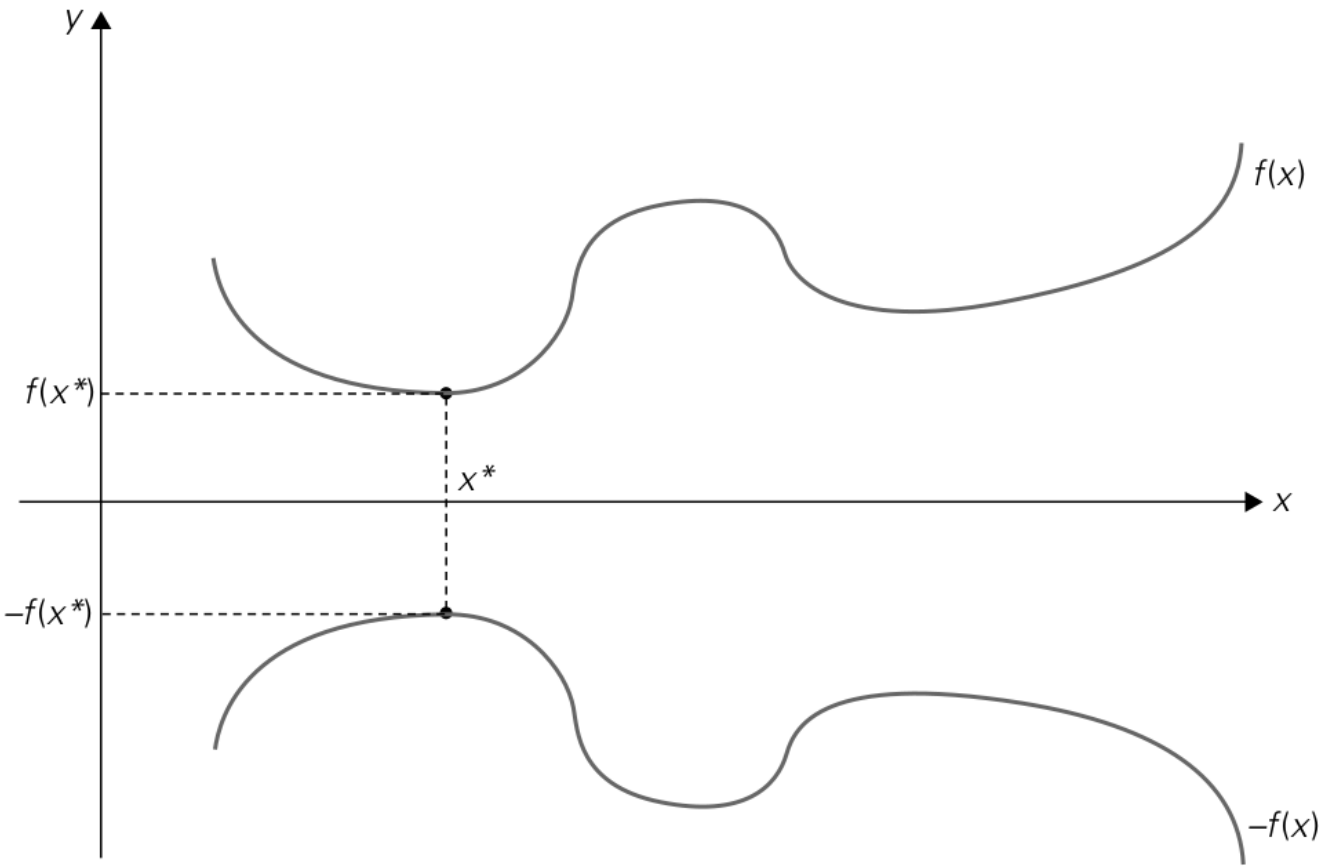


Figure: Hubungan antara masalah minimalisasi dan maksimalisasi: $\min f(x) = -\max[-f(x)]$

4. Pemrograman linier (1)

Pemrograman linier: masalah optimalisasi fungsi linier dari beberapa variabel yang dengan batasan (kendala) dalam bentuk *persamaan linier* dan *pertidaksamaan linier*.

- Banyak masalah pengambilan keputusan yang optimal dapat direduksi menjadi sebuah contoh dari masalah Pemrograman linier.

Contoh

Diketahui dana abadi universitas yang memiliki nilai investasi sebesar $100 juta. Jumlah ini harus dibagi antara tiga jenis investasi: saham, obligasi, dan uang tunai. Manajer dana abadi mengharapkan pengembalian tahunan sebesar 10%, 7%, *dan* 3% *masing-masing untuk investasi saham, obligasi, dan tunai.*

Karena saham lebih berisiko daripada obligasi, aturan endowmen mensyaratkan jumlah yang diinvestasikan dalam saham tidak lebih dari sepertiga uang yang diinvestasikan dalam obligasi. Selain itu, setidaknya 25% *dari jumlah total investasi saham dan obligasi harus diinvestasikan dalam bentuk tunai.*

Bagaimana seharusnya para manajer menginvestasikan uang untuk memaksimalkan pengembalian?

4. Pemrograman linier (2)

Model matematika dari masalah ini: Misalkan x, y, dan z adalah jumlah (dalam jutaan dolar) yang diinvestasikan masing-masing dalam saham, obligasi, dan uang tunai.

$$\begin{aligned} \text{maximize} \quad & 0.10x + 0.07y + 0.03z \\ \text{subject to} \quad & x + y + z = 100 \\ & x \leq \frac{1}{3}y \\ & z \geq 0.25(x + y) \\ & x \geq 0,\ y \geq 0,\ z \geq 0 \end{aligned}$$

Definisi (Formulasi umum program linier)

$$\begin{aligned} \textit{maksimalkan (atau minimalkan)} \quad & c_1x_1 + \cdots + c_nx_n \\ \textit{dengan kendala} \quad & a_{i1x_1} + \cdots + a_{in}x_n \leq b_i,\ \textit{untuk } i = 1, \ldots, m \\ & \textit{(dalam hal ini } \leq \textit{ dapat diganti dengan } \geq \textit{ atau } =) \\ & x_1 \geq 0, \ldots,\ x_n \geq 0 \end{aligned}$$

end of slide...