08 - Transform and Conquer

[KOMS119602] & [KOMS120403]

Design and Analysis of Algorithm (2021/2022)

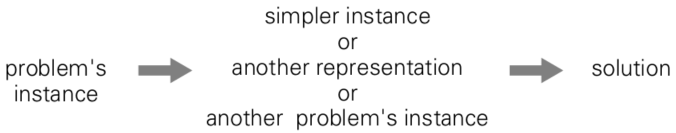
Dewi Sintiari

Prodi S1 Ilmu Komputer
Universitas Pendidikan Ganesha

Week 21-25 March 2022

Table of contents

Principal of Transform-and-Conquer (1)



Principal of Transform-and-Conquer (1)

Three major variations:

1. Instance simplification: transformation to a simpler or more convenient instance of the same problem.
2. Representation change: transformation to a different representation of the same instance.
3. Problem reduction: transformation to an instance of a different problem for which an algorithm is already available.

Instance simplification

1. Presorting (1)

- Many questions about a list/array are easier to answer if the list is sorted.
- The time efficiency of algorithms that involve sorting may depend on the efficiency of the sorting algorithm being used.

Recall sorting algorithms:

- Selection sort, bubble sort, and insertion sort: complexity $\Theta(n^2)$.
- Merge sort and Quick sort: complexity $\Theta(n \log n)$ in the average case but is quadratic in the worst case.

1. Presorting (2): example

Example (Checking element uniqueness in an array)

Problem: Given an array, check if there exist equal elements.

- Brute-force:

  Compare pairs of the arrays elements until either two equal elements were found or no more pairs were left. Complexity: $\Theta(n^2)$.

- With presorting:

  sort the array first and then **check only the consecutive elements**: if the array has equal elements, a pair of them must be next to each other, and vice versa.

1. Presorting (2): pseudocode

Checking uniqueness in an array with presorting

Algorithm 1 Checking uniqueness in an array

```
procedure UNIQUE($A[0..n-1]$)
    input: an ordorable array $A[0..n-1]$
    output: "True" if $A$ has no equal elements, "False" otherwise
    Sort the array $A$                              ▷ Implement the presorting
    for $i \leftarrow 0$ to $n-2$ do
        if $A[i] = A[i+1]$ then                     ▷ Compare the adjacent elements
            return false
        end if
    end for
    return true
end procedure
```

1. Presorting (2): time complexity

TC = time on sorting + time on checking consecutive elements

- Time for checking consecutive elements: $\leq n - 1$

So the TC depends on the sorting algorithm used in presorting

- If use Selection sort, etc., then TC in $\Theta(n^2)$
- If use Merge sort/Quick sort, then TC in $\Theta(n \log n)$

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

This gives a better complexity than brute-force approach.

2. Searching problem

- **Without sorting**: Sequential search (brute-force), complexity $\Theta(n)$
- **With sorting**: binary search (recursive, needs sorted array)

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

So, in this case, Sequential Search is the better option.

Representation change

1. Gaussian elimination

Systems of two linear equations (you are already familiar with):

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

where: a_{ij} and b_i for $i, j \in \{1, 2\}$ is a real number; x and y are unknown variables.

Problem

Given a system of n equations in n unknown variables:

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

The task is to solve the system, i.e. find the values $[x_1, x_2, \ldots, x_n]$

1. Gaussian elimination

Obvious algorithm: use repeated substitution/elimination as in the case $n = 2$.

More elegant way: 3. Gaussian elimination

- The idea is to transform a system of n linear equations in n unknowns to an equivalent system (i.e., a system with the same solution as the original one) with an upper- triangular coefficient matrix, a matrix with all zeros below its main diagonal.

We aim to transfer it into:

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

1. Gaussian elimination

In matrix notations, this can be written as:

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

where:

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

Why is the system with the upper-triangular coefficient matrix better than a system with an arbitrary coefficient matrix?

- Because we can easily solve the system with an upper-triangular coefficient matrix by back substitutions.

1. Gaussian elimination

Solving the linear system

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

- First, we can immediately find the value of x_n from the last equation, namely: $a'_{nn}x_n = b'_n$.
- Then we can substitute this value into the next to last equation, namely $a'_{(n-1)(n-1)}x_{n-1} + a'_{(n-1)n}x_n = b'_n$ to get x_{n-1}.
- ...and so on, until we substitute the known values of the last $n-1$ variables into the first equation, from which we find the value of x_1.

1. Gaussian elimination

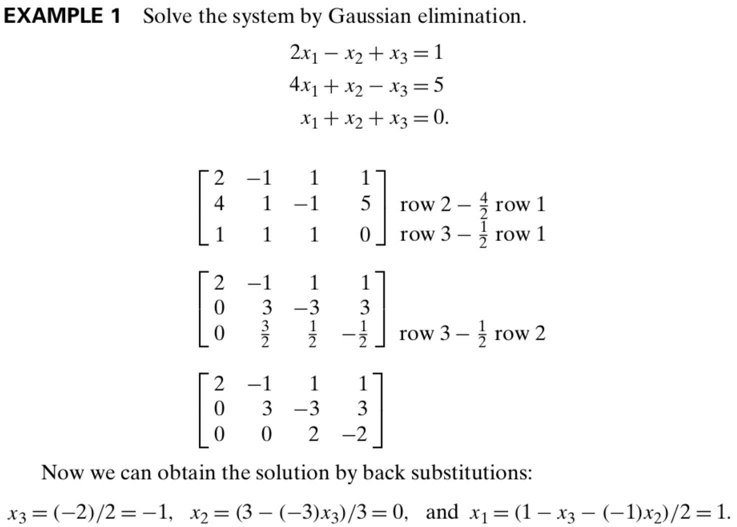
How can we get from a system with an arbitrary coefficient matrix A to an equivalent system with an upper-triangular coefficient matrix A?

We can do that through a series of (the so-called) elementary operations.

- exchanging two equations of the system;
- replacing an equation with its nonzero multiple;
- replacing an equation with a sum or difference of this equation and some multiple of another equation

1. Gaussian elimination

Example: Implementation of Gaussian elimination (A. Levitin, page 236)



EXAMPLE 1 Solve the system by Gaussian elimination.

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

$$\begin{array}{l} \left[\begin{array}{rrrr} 2 & -1 & 1 & 1 \\ 4 & 1 & -1 & 5 \\ 1 & 1 & 1 & 0 \end{array}\right] \begin{array}{l} \\ \text{row } 2 - \frac{4}{2} \text{ row } 1 \\ \text{row } 3 - \frac{1}{2} \text{ row } 1 \end{array} \end{array}$$

$$\begin{array}{l} \left[\begin{array}{rrrr} 2 & -1 & 1 & 1 \\ 0 & 3 & -3 & 3 \\ 0 & \frac{3}{2} & \frac{1}{2} & -\frac{1}{2} \end{array}\right] \begin{array}{l} \\ \\ \text{row } 3 - \frac{1}{2} \text{ row } 2 \end{array} \end{array}$$

$$\left[\begin{array}{rrrr} 2 & -1 & 1 & 1 \\ 0 & 3 & -3 & 3 \\ 0 & 0 & 2 & -2 \end{array}\right]$$

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. Gaussian elimination

For the pseudocode of the algorithm and computing the time complexity, see the next slides (this is not discussed in class).

Read **page 236-238** of the book A. Levitin for an explanation!

1. Gaussian elimination

Here is pseudocode of the first stage, called forward elimination, of the algorithm.

```
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 $i \leftarrow 1$ to $n$ do $A[i, n + 1] \leftarrow b[i]$   //augments the matrix
    for $i \leftarrow 1$ to $n - 1$ do
        for $j \leftarrow i + 1$ to $n$ do
            for $k \leftarrow i$ to $n + 1$ do
                $A[j, k] \leftarrow A[j, k] - A[i, k] * A[j, i] / A[i, i]$
```

1. Gaussian elimination

An improvement of the previous algorithm (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. Gaussian elimination

Time complexity of "*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. Gaussian elimination for LU-decomposition

Other examples of "Instance simplification"

- Lower-Upper (LU) Decomposition: factor a matrix as the product of a lower triangular matrix and an upper triangular matrix. Example:

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

How is this decomposition useful?

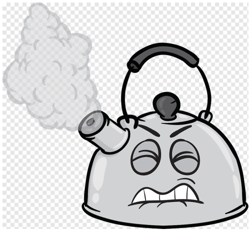
- To solve the linear system $Ax = b$, we can decompose A into $A = LU$, so $Ax = b \Leftrightarrow LUx = b$
- Now assume $Ux = y$, so $LUx = b \Leftrightarrow Ly = b$
- Solve the linear system $Ly = b$, so we obtain y.
- Then solve the linear system $Ux = y$, so we obtain x.
- Here, solving $Ly = b$ or $Ux = y$ can be done easily by substitution, because L is a lower-triangular matrix and U is an upper-triangular matrix.

Problem reduction

Problem reduction

A mathematician joke of problem reduction

Professor X, a noted mathematician, noticed that when his wife wanted to boil water for their tea, she took their kettle from their cupboard, filled it with water, and put it on the stove. Once, when his wife was away, the professor had to boil water by himself. He saw that the kettle was sitting on the kitchen counter. What did Professor X do? He put the kettle in the cupboard first and then proceeded to follow his wifes routine.



Problem reduction

Problem reduction: If you need to solve a problem, reduce it to another problem that you know how to solve.

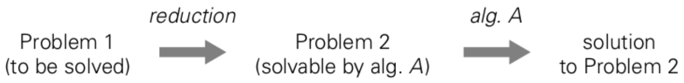


Figure: Problem reduction strategy

1. Computing the Least Common Multiple (1)

The least common multiple of two positive integers m and n, denoted $\mathrm{lcm}(m,n)$, is defined as the smallest integer that is divisible by both m and n.

Example: $\mathrm{lcm}(24,60) = 120$, and $\mathrm{lcm}(11,5) = 55$.

Middle-school method for computing lcm:

- Find the prime factorizations of m and n;
- Compute the product of all the common prime factors of m and n;
- Multiply with all the prime factors of m that are not in n, and all the prime factors of n that are not in m.

Example:

$$24 = 2 \cdot 2 \cdot 2 \cdot 3$$
$$60 = 2 \cdot 2 \cdot 3 \cdot 5$$
$$\mathrm{lcm}(24,60) = (2 \cdot 2 \cdot 3) \cdot 2 \cdot 5 = 120$$

1. Computing the Least Common Multiple (2)

Note that gcd(m, n) can be computed by taking product of *all common prime divisors of m and n*.

Example:

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

We can see that the following relation holds:

$$m \cdot n = \gcd(m, n) \cdot \operatorname{lcm}(m, n) \Leftrightarrow \operatorname{lcm}(m, n) = \frac{m \cdot n}{\gcd(m, n)}$$

So, the problem of finding lcm is reduced to finding gcd, and this can be solved using Euclidean gcd algorithm.

2. Counting Paths in a Graph

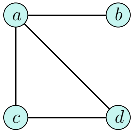
Problem: Given a graph G, and vertices v_i and v_j in G. Find the number of paths in G from v_i to v_j.

Lemma (The number of different paths)

Let G be an undirected graph with adjacency matrix A, w.r.t. the ordering $v_1, v_2, \ldots, v_n$ of $V(G)$. The number of different paths of length $r > 0$ from v_i to v_j, equals to the (i,j)-th entry of A^r.

So, this problem reduced to *computing the power of square matrices*.

Example:



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

Example: the number of paths of length 2 from a to c equals to the entry (a, c) of matrix A^2, namely 1 (the only path is $a - d - c$).

3. Reduction of Optimization Problems (1)

- Maximization problem: a problem that asks to find a maximum of some function.
- Minimization problem: a problem that asks to find a minimum of some function.

Suppose now that you need to find *a minimum of some function* $f(x)$ and you have an algorithm for function maximization. How can you take advantage of the latter?

We have relation

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

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

3. Reduction of Optimization Problems (2)

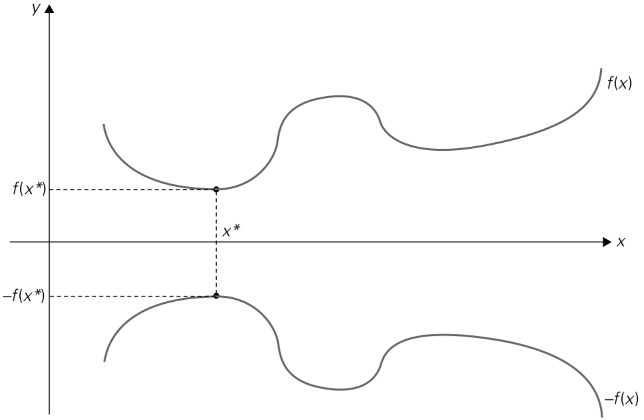


Figure: Relationship between minimization and maximization problems: $\min f(x) = -\max[-f(x)]$

4. Linear programming (1)

Linear programming: a problem of optimizing a linear function of several variables subject to constraints in the form of linear equations and linear inequalities.

- Many problems of optimal decision making can be reduced to an instance of the linear programming problem.

Example

Consider a university endowment that needs to invest \$100 million. This sum has to be split between three types of investments: stocks, bonds, and cash. The endowment managers expect an annual return of 10%, 7%, and 3% for their stock, bond, and cash investments, respectively. Since stocks are more risky than bonds, the endowment rules require the amount invested in stocks to be no more than one-third of the moneys invested in bonds. In addition, at least 25% of the total amount invested in stocks and bonds must be invested in cash. How should the managers invest the money to maximize the return?

4. Linear programming (2)

A mathematical model of this problem: Let x, y, and z be the amounts (in millions of dollars) invested in stocks, bonds, and cash, respectively.

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

Definition (General formulation of linear program)

$$\begin{aligned} \text{maximize (or minimize)} \quad & c_1x_1 + \cdots + c_nx_n \\ \text{subject to} \quad & a_{i1x_1} + \cdots + a_{in}x_n \leq b_i, \text{ for } i = 1, \ldots, m \\ & (\textit{Here, } \leq \textit{ can be replaced with } \geq \textit{ or } =) \\ & x_1 \geq 0, \ldots,\ x_n \geq 0 \end{aligned}$$