CSC 411
Design and Analysis of Algorithms

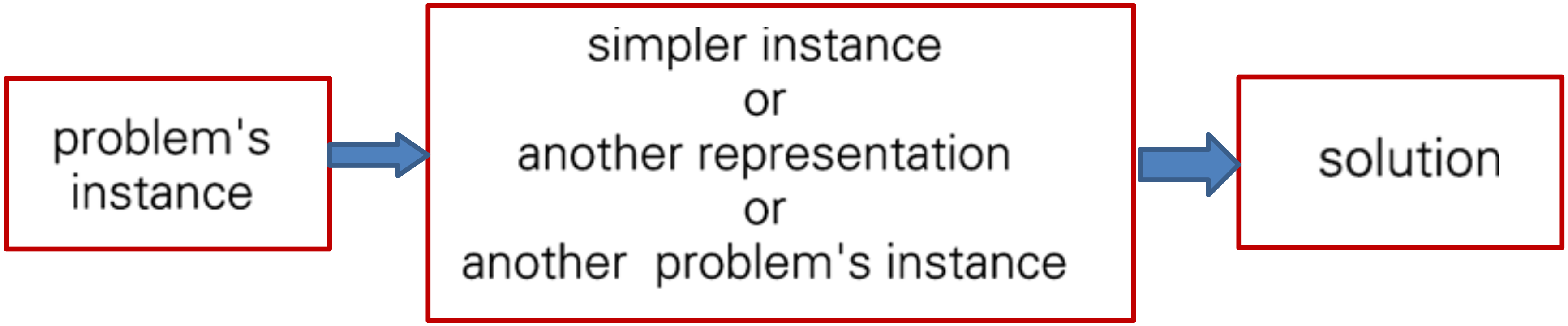
Chapter 6 Transfer and Conquer
- Part 1

Instructor: Minhee Jun

Transform and Conquer

- This group of techniques solves a problem by a ***transformation***
  - to a simpler/more convenient instance of the same problem (*instance simplification*)
  - to a different representation of the same instance (*representation change*)
  - to a different problem for which an algorithm is already available (*problem reduction*)

Transform and Conquer strategy



Transfer-and-Conquer Examples

- Presorting (6.1)
- Gaussian Elimination (6.2)
- Balanced Search Trees (6.3)
    - AVL Trees
    - 2-3 Trees
- Heaps and Heapsort (6.4)
- Horner’s Rule and Binary Exponentiation (6.5)

6.1 Presorting

- Instance simplification
  - Solve a problem's instance by transforming it into another simpler or easier instance of the same problem
- Presorting
  - Many problems involving lists are easier when list is sorted.
    - searching
    - computing the median (selection problem)
    - checking if all elements are distinct (element uniqueness)
  - Also:
    - Topological sorting helps solving some problems for dags.
    - Presorting is used in many geometric algorithms.

How fast can we sort ?

- Efficiency of algorithms involving sorting depends on efficiency of sorting.
  - Selection sort, bubble sort, insertion sort:
    - $\Theta(n^2)$ in the worst and average cases.
  - Merge sort:
    - $\Theta(n \log n)$ always
  - Quick sort:
    - $\Theta(n \log n)$ in the average cases, $\Theta(n^2)$ in the worst
- No general comparison-based sorting algorithm can have a better efficiency than $\Theta(n \log n)$ in the worst case or average cases (Section 11.2)

Example 1: Finding Element Uniqueness

- Given a list *A* of *n* orderable elements, determine if there are any duplicates of any element

Brute Force:

```
for each x ∈ A
  for each y ∈ {A – x}
      if x = y return false
return true
```

Presorting:

```
Sort A
for i ← 1 to n-1
   if A[i] = A[i+1] return false
return true
```

Runtime?

Efficiency of Element Uniqueness Algorithms

- Brute force algorithm
  - Compare all pairs of elements
  - Efficiency: $O(n^2)$

- Presorting-based algorithm
  - Stage 1: sort by efficient sorting algorithm (e.g. mergesort)
  - Stage 2: scan array to check pairs of adjacent elements
  - Efficiency: $\Theta(n \log n) + O(n) = \Theta(n \log n)$

- Another algorithm?
  - Hashing (do you own research if you are interested.)

Example 2: Searching with presorting

- Problem: Search for a given *K* in A[0..*n*-1]
- Presorting-based algorithm:
  - Stage 1 Sort the array by an efficient sorting algorithm
  - Stage 2 Apply binary search
- **Efficiency?**
  - $\Theta(n \log n) + O(\log n) = \Theta(n \log n)$
- Good or bad?
  - Why do we have our dictionaries, telephone directories, etc. sorted?
  - If we are to search in the same list more than once, the time spent on sorting is justified.

6.2 Gaussian Elimination

- **Given:** A system of *n* linear equations in *n* unknowns with an arbitrary coefficient matrix.
- **Transform to:**
  - An equivalent system of *n* linear equations in *n* unknowns with an upper triangular coefficient matrix.
  - An example of transform and conquer through **representation change**
- Backward substitution
  - Solve the latter by substitutions starting with the last equation and moving up to the first one.

Gaussian Elimination

- Consider a system of two linear equations:

$$a_{11}x + a_{12}y = b_1$$
$$a_{21}x + a_{22}y = b_2.$$

- To solve this we can rewrite the first equation to solve for x:

$$a_{11}x = (b_1 - a_{12}y) \quad \rightarrow \quad x = (b_1 - a_{12}y)/a_{11}$$

- And then substitute x in the second equation to solve for y. After we solve for y, we can then solve for x:

$$a_{21} \cdot (b_1 - a_{12}y)/a_{11} + a_{22}y = b_2 \rightarrow \quad y = ?$$

Gaussian Elimination

- In many applications we need to solve a system of ***n*** equations with ***n*** unknowns, e.g.:

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

- If ***n*** is a large number it is very cumbersome to solve these equations using the substitution method.

Gaussian Elimination

- **Gaussian elimination**
  - Fortunately there is a more elegant algorithm to solve such systems of linear equations
  - The idea is to transform the system of linear equations into an equivalent one that **eliminates coefficients** so we end up with a triangular matrix.

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
&\vdots \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n
\end{aligned}
\quad \Longrightarrow \quad
\begin{aligned}
a'_{11}x_1 + a'_{12}x_2 + \cdots + a'_{1n}x_n &= b'_1 \\
a'_{22}x_2 + \cdots + a'_{2n}x_n &= b'_2 \\
&\vdots \\
a'_{nn}x_n &= b'_n.
\end{aligned}
$$

Gaussian Elimination

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
&\vdots \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n
\end{aligned}
\quad \Longrightarrow \quad
\begin{aligned}
a'_{11}x_1 + a'_{12}x_2 + \cdots + a'_{1n}x_n &= b'_1 \\
a'_{22}x_2 + \cdots + a'_{2n}x_n &= b'_2 \\
&\vdots \\
a'_{nn}x_n &= b'_n.
\end{aligned}$$

- The matrix with zeros in the lower triangle (it is called an **upper triangular matrix**) is easier to solve.
- We can solve the last equation first, substitute into the second to last, etc. working our way back to the first one.

Pseudocode of Gaussian Elimination

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**
      $A[j, k] \leftarrow A[j, k] - A[i, k] * A[j, i] \,/\, A[i, i]$

Example: Gaussian Elimination

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

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

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

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

Efficiency of Gaussian Elimination

Efficiency: $\Theta(n^3) + \Theta(n^2) = \Theta(n^3)$

- Stage 1: Reduction to the upper-triangular matrix

  $\Theta(n^3)$

- Stage 2: Backward substitution

  $\Theta(n^2)$

LU Decomposition

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

$$A = \begin{bmatrix} 2 & -1 & 1 \\ 4 & 1 & -1 \\ 1 & 1 & 1 \end{bmatrix}.$$

$$A = L \cdot U$$

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

$$Ax = b \rightarrow L \cdot Ux = b$$

$$y = Ux \rightarrow Ly = b$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ \frac{1}{2} & \frac{1}{2} & 1 \end{bmatrix}$$

Example: LU Decomposition

$$A = L \cdot U \qquad Ax = b \to L \cdot Ux = b \qquad y = Ux \to Ly = b$$

$$A = \begin{bmatrix} 2 & -1 & 1 \\ 4 & 1 & -1 \\ 1 & 1 & 1 \end{bmatrix}.$$

1. $Ly = b$

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ \frac{1}{2} & \frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 0 \end{bmatrix}$$

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

$$y_1 = 1, \qquad y_2 = 5 - 2y_1 = 3, \qquad y_3 = 0 - \frac{1}{2}y_1 - \frac{1}{2}y_2 = -2.$$

1. $y = Ux$

$$\begin{bmatrix} 2 & -1 & 1 \\ 0 & 3 & -3 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ -2 \end{bmatrix},$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ \frac{1}{2} & \frac{1}{2} & 1 \end{bmatrix}$$

$$x_3 = (-2)/2 = -1,$$

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