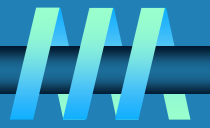
CS 395 – Analysis of Algorithms

Chapter 6 – Transform-and-Conquer

Read 6.1 & 6.2

- **Transform-and-Conquer Technique**
  - **Basic Idea**
  - **Presorting**
  - **Gaussian Elimination**

Transform-and-Conquer



Two-stage procedure:

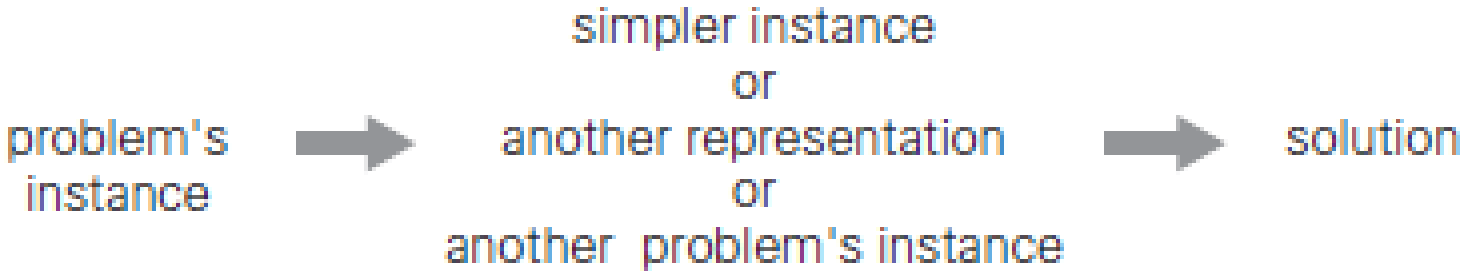
1. **Transformation stage: problem's instance modified to be more amenable to solution.**
2. **Conquering stage: the problem is solved**

Three Transformation variations:

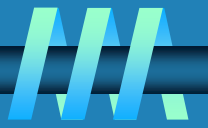
1. **a simpler/more convenient instance of the same problem (*instance simplification*)**
2. **a different representation of the same instance (*representation change*)**
3. **a different problem for which an algorithm is already available (*problem reduction*)**

How does a mathematician boil water in a kitchen that is on fire?

Transform-and-Conquer strategy



Instance simplification - Presorting



Solve a problem's instance by transforming it into another simpler/easier instance of the same problem

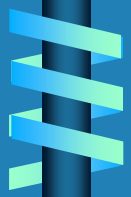
<u>Presorting</u>

Many problems involving lists are easier when list is sorted, e.g.

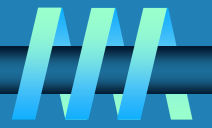
- searching
- computing the median (selection problem)
- checking if all elements are distinct (element uniqueness)

Also:

- Topological sorting helps solving some problems for DAGs.
- Presorting is used in many geometric algorithms.



Searching with presorting



Problem: Search for a given K in A[0..n-1]

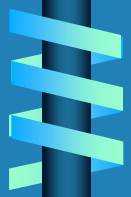
Presorting-based algorithm:

- **Stage 1 Sort the array by an efficient sorting algorithm**
- **Stage 2 Apply binary search**

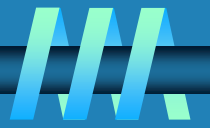
Efficiency: $\Theta(n \log n) + O(\log n) = \Theta(n \log n)$

Good or bad?

Why do we have our dictionaries, telephone directories, etc. sorted?



Element Uniqueness with presorting



- **Presorting-based algorithm**

  Stage 1: sort by efficient sorting algorithm (e.g. mergesort)

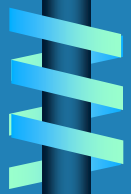
  Stage 2: scan array to check pairs of adjacent elements

  Efficiency: $\Theta(n \log n) + O(n) = \Theta(n \log n)$

- **Brute force algorithm**

  Compare all pairs of elements

  Efficiency: $O(n^2)$



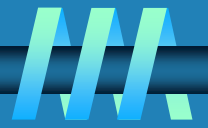
Instance simplification – Gaussian Elimination

- 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.

  Solve the latter by substitutions starting with the last equation and moving up 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**
            $A[j, k] \leftarrow A[j, k] - A[i, k] * A[j, i] / A[i, i]$

Can you see the issue here?

Instance simplification – Gaussian Elimination

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

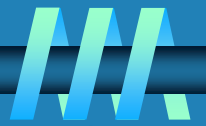
In matrix notations, we can write this as

$$Ax = b \quad \Longrightarrow \quad A'x = b',$$

where

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

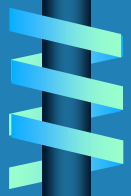
Gaussian Elimination (cont.)



The transformation is accomplished by a sequence of elementary operations on the system's coefficient matrix (which don't change the system's solution):

for ***i*** ←**1 to *n*-1** do

replace each of the subsequent rows (i.e., rows *i*+1, …, *n*) by a difference between that row and an appropriate multiple of the *i*-th row to make the new coefficient in the *i*-th column of that row 0



Example of Gaussian Elimination

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

Stage 1: Forward Elimination

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

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

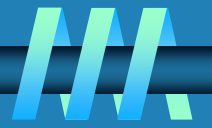
$$
\begin{bmatrix} 2 & -1 & 1 & 1 \\ 0 & 3 & -3 & 3 \\ 0 & 0 & 2 & -2 \end{bmatrix}
$$

Stage 2: Back Substitution

Now we can obtain the solution by back substitutions:

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

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

What if A[i, i]= 0

Θ(n3)

How often does A[j, i]/A[i, i] change?
Could we avoid this expensive operation?

Pseudocode of Gaussian Elimination

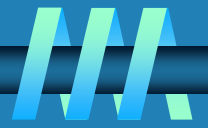
ALGORITHM *BetterForwardElimination*$(A[1..n, 1..n], b[1..n])$

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

Partial Pivoting

More efficient

Pseudocode of Gaussian Elimination



Stage 2: Back substitutions

for $j \leftarrow n$ downto 1 do

    $t \leftarrow 0$

    for $k \leftarrow j+1$ to n do

        $t \leftarrow t + A[j, k] * x[k]$

    $x[j] \leftarrow (A[j, n+1] - t) / A[j, j]$

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

Forward Elimination

Back Substitution