

## Chapter 6: Transform and Conquer

You can never solve a problem on the level on which it was created. (Albert Einstein)

We cannot solve our problems with the same thinking we used when we created them. (Albert Einstein)

Introduction . . . . .	2
Presorting Example . . . . .	3
<b>Gaussian Elimination</b> . . . . .	<b>4</b>
System of Linear Equations . . . . .	4
Example of Linear Equations . . . . .	5
Forward Elimination . . . . .	6
Forward Elimination Algorithm . . . . .	7
Backward Substitution . . . . .	8
Backward Substitution Algorithm . . . . .	9
Comments on Gaussian Elimination. . . . .	10
<b>Balanced Search Trees: AVL Trees</b> . . . . .	<b>11</b>
Binary Search Tree Notation . . . . .	11
AVL Trees . . . . .	12
Binary Search Tree Insertion . . . . .	13
R-Rotation . . . . .	14
LR-Rotation . . . . .	15

Balancing AVL Trees. . . . .	16
AVL Illustration . . . . .	17
Analysis of AVL Trees . . . . .	18
<b>Horner's Rule and Fast Exponentiation</b> . . . . .	<b>19</b>
Horner's Rule. . . . .	19
Horner's Algorithm . . . . .	20
Fast Exponentiation . . . . .	21
Fast Exponentiation Algorithm . . . . .	22

## Introduction

*Transform-and-conquer* is an approach to solving a problem by changing an instance to:

- a simpler instance of the same problem, or
- a different representation of the same problem, or
- an instance of a different problem.

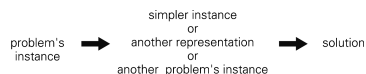


FIGURE 6.1 Transform-and-conquer strategy  
CS 3343 Analysis of Algorithms

Chapter 6: Slide – 2

## Presorting Example

Presorting (sorting the data in advance) makes it easier to:

- Check element uniqueness.
- Compute the mode.
- Search the data.
- Find the minimum, maximum, median, or any order statistic.

CS 3343 Analysis of Algorithms

Chapter 6: Slide – 3

## Gaussian Elimination

### System of Linear Equations

*Gaussian Elimination* is an algorithm solving a system of  $n$  linear equations in  $n$  unknowns.

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

where  $a_{ij}$  and  $b_i$  are the coefficients, and  $x_j$  are the unknowns.

For example, the linear least squares method in statistics needs to solve this kind of system.

CS 3343 Analysis of Algorithms

Chapter 6: Slide – 4

### Example of Linear Equations

For example, consider this system of 3 equations:

$$\begin{aligned}2x_1 - x_2 + 3x_3 &= 4 \\3x_1 + 2x_2 + x_3 &= 6 \\-5x_1 + x_2 - 2x_3 &= 3\end{aligned}$$

This system has the solution  $x_1 = -5/4$ ,  $x_2 = 13/4$ , and  $x_3 = 13/4$ .

Gaussian elimination works in two stages: forward elimination and backward substitution.

Forward elimination adds/subtracts equations from each other.

CS 3343 Analysis of Algorithms

Chapter 6: Slide – 5

## Forward Elimination

$$\begin{bmatrix} 2 & -1 & 3 & 4 \\ 3 & 2 & 1 & 6 \\ -5 & 1 & -2 & 3 \end{bmatrix} \begin{array}{l} \text{row 2} - 3/2 \text{ row 1} \\ \text{row 3} + 5/2 \text{ row 1} \end{array}$$

$$\begin{bmatrix} 2 & -1 & 3 & 4 \\ 0 & 7/2 & -7/2 & 0 \\ 0 & -3/2 & 11/2 & 13 \end{bmatrix} \text{row 3} + 3/7 * \text{row 2}$$

$$\begin{bmatrix} 2 & -1 & 3 & 4 \\ 0 & 7/2 & -7/2 & 0 \\ 0 & 0 & 4 & 13 \end{bmatrix}$$

CS 3343 Analysis of Algorithms

Chapter 6: Slide – 6

## Forward Elimination Algorithm

```
algorithm ForwardElim( $A[1..n, 1..n], b[1..n]$ )
// First stage to solve  $Ax = b$  by Gaussian Elim.
// Input: A matrix  $A$  and column vector  $b$ 
// Output: Equivalent upper-triangular matrix
for  $i \leftarrow 1$  to  $n$  do
   $A[i, n+1] \leftarrow b[i]$ 
  for  $i \leftarrow 1$  to  $n-1$  do
    for  $j \leftarrow i+1$  to  $n$  do
      for  $k \leftarrow i$  to  $n+1$  do
         $A[j, k] \leftarrow A[j, k] - A[j, i] * A[i, k] / A[i, i]$ 
```

CS 3343 Analysis of Algorithms

Chapter 6: Slide – 7

## Backward Substitution

Solving last equation  $4x_3 = 13$  yields  $x_3 = 13/4$ .

Substitute for  $x_3$  in second equation:

$$\begin{aligned} 7/2x_2 - 7/2x_3 &= 0 \\ 7/2x_2 &= 0 + 7/2x_3 = 0 + 91/8 = 91/8 \\ x_2 &= (91/8)/(7/2) = 13/4 \end{aligned}$$

Substitute for  $x_2$  and  $x_3$  in first equation:

$$\begin{aligned} 2x_1 - x_2 + 3x_3 &= 4 \\ 2x_1 &= 4 + x_2 - 3x_3 = 4 + 13/4 - 3 * (13/4) \\ 2x_1 &= -10/4 \\ x_1 &= -5/4 \end{aligned}$$

CS 3343 Analysis of Algorithms

Chapter 6: Slide – 8

## Backward Substitution Algorithm

```
algorithm BackwardSubst( $A[1..n, 1..n+1]$ )
// Second stage to solve  $Ax = b$  by Gaussian Elim.
// Input: An upper-triangular matrix  $A$ 
// Output: Solution column-vector  $x$ 
for  $i \leftarrow n$  downto 1 do
   $s \leftarrow A[i, n+1]$ 
  for  $j \leftarrow i+1$  to  $n$  do
     $s \leftarrow s - A[i, j] * x[j]$ 
   $x[i] \leftarrow s / A[i, i]$ 
return  $x$ 
```

CS 3343 Analysis of Algorithms

Chapter 6: Slide – 9

## Comments on Gaussian Elimination

- Gaussian Elimination requires that  $A$  be nonsingular. See your linear algebra book.
- Gaussian Elimination is  $\Theta(n^3)$ . Note triple loop in *ForwardElim*.
- Better numerical properties can be obtained through *partial pivoting*. See book.
- *LU decomposition* is an extension of Gaussian elimination. After  $\Theta(n^3)$  preprocessing of the  $A$  matrix, only  $\Theta(n^2)$  is needed for any  $b$  vector. See book.
- Similar processing can be used to compute matrix inverse and determinant.

CS 3343 Analysis of Algorithms

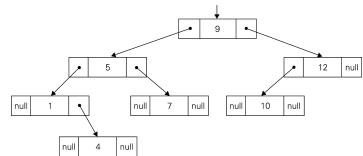
Chapter 6: Slide – 10

## Balanced Search Trees: AVL Trees

11

### Binary Search Tree Notation

- $T.root$  is the root of tree  $T$  or **null**.
- $x.key$  is the key of node  $x$ .
- $x.parent$  is the parent of  $x$  or **null**.
- $x.left$  is the left child of  $x$  or **null**.
- $x.right$  is the right child of  $x$  or **null**.
- $x.height$  is the height of  $x$ . **null.height** =  $-1$ .



CS 3343 Analysis of Algorithms

Chapter 6: Slide – 11

## AVL Trees

- For each node in binary search tree, all keys on its left are  $\leq$  to its key, and all keys on its right are  $\geq$  to its key.
- In a balanced search tree, the height is  $\Theta(\log n)$ , where  $n$  is the number of nodes.
- For each node in an AVL tree, the heights of the left and right subtree are equal or differ by one.
- The balance factor is the left height minus the right height.
- Tree *rotations* are used to transform the tree into balance.

CS 3343 Analysis of Algorithms

Chapter 6: Slide – 12

## Binary Search Tree Insertion

```
algorithm TreeInsert( $T, z$ )
// Inserts a node into a binary search tree
// Input: Tree  $T$  and node  $z$ 
 $y \leftarrow \text{null}$ 
 $x \leftarrow T.root$ 
while  $x \neq \text{null}$  do
     $y \leftarrow x$ 
    if  $z.key < x.key$  then  $x \leftarrow x.left$ 
    else  $x \leftarrow x.right$ 
 $z.parent \leftarrow y$ 
if  $y = \text{null}$  then  $T.root \leftarrow z$ 
else if  $z.key < y.key$  then  $y.left \leftarrow z$ 
else  $y.right \leftarrow z$ 
```

CS 3343 Analysis of Algorithms

Chapter 6: Slide – 13

## R-Rotation

An R-rotation is used when the left child is too high, and the left-left grandchild is as high or higher than the left-right one.

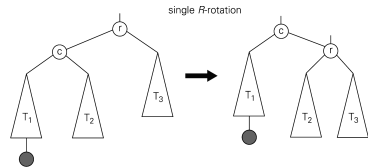


FIGURE 6.4 General form of the R-rotation in the AVL tree. A shaded node is the last one inserted.

CS 3343 Analysis of Algorithms

Chapter 6: Slide – 14

## LR-Rotation

An LR-rotation is used when the left child is too high and the left-right grandchild is higher than the left-left one.

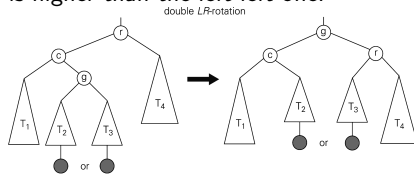


FIGURE 6.5 General form of the double LR-rotation in the AVL tree. A shaded node is the last one inserted. It can be either in the left subtree or in the right subtree of the root's grandchild.

CS 3343 Analysis of Algorithms

Chapter 6: Slide – 15

## Balancing AVL Trees

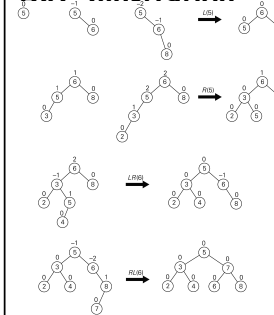
```

algorithm AVLBalance(x)
  // Input: Node x
  // Output: x and nodes above it are balanced
  while x ≠ null do
    if x.left.height > x.right.height + 1 then
      if x.left.left.height < x.left.right.height
        then LR-Rotation(x)
      else R-Rotation(x)
    else if x.left.height + 1 < x.right.height then
      if x.right.left.height > x.right.right.height
        then RL-Rotation(x)
      else L-Rotation(x)
    x ← x.parent
  
```

CS 3343 Analysis of Algorithms

Chapter 6: Slide – 16

## AVL Illustration



CS 3343 Analysis of Algorithms

Chapter 6: Slide – 17

## Analysis of AVL Trees

The height  $h$  of an AVL tree with  $n$  nodes satisfies  $\log_2(n/2) < h < 1.5 \log_2 n$

- A binary tree of height  $h$  has  $\leq 2^{h+1} - 1$  nodes.  
 $n < 2^{h+1} \rightarrow n/2 < 2^h \rightarrow \log_2(n/2) < h$
- An AVL tree with height  $h$  at worst has subtrees of height  $h - 1$  and  $h - 2$ .
- Assuming heights  $h - 1$  and  $h - 2$  have at least  $1.6^{h-1}$  and  $1.6^{h-2}$  nodes, respectively:  
 $n > 1.6^{h-1} + 1.6^{h-2} > 1.6^h \rightarrow$   
 $1.6^h < n \rightarrow h < 1.5 \log_2 n$

CS 3343 Analysis of Algorithms

Chapter 6: Slide - 18

## Horner's Rule and Fast Exponentiation

19

### Horner's Rule

Horner's rule is an efficient algorithm for evaluating a polynomial. For example,

- $5x^2 - 3x + 8$  can be evaluated by  $(5x + 3)x + 8$ .
- $7x^3 + x^2 - 9x - 2$  can be evaluated by  $((7x + 1)x - 9)x - 2$ .

The generalization of this is Horner's rule.

It performs  $n$  multiplications and  $n$  additions/subtractions for an  $n$ -degree polynomial.

CS 3343 Analysis of Algorithms

Chapter 6: Slide - 19

## Horner's Algorithm

```
algorithm Horner( $P[0..n], x$ )
// Evaluates a polynomial at  $x$ 
// Input: A value  $x$  and an array  $P$  of coefficients
// Output: The value of the polynomial at  $x$ 
 $v \leftarrow P[n]$ 
for  $i \leftarrow n - 1$  downto  $0$  do
     $v \leftarrow x * v + P[i]$ 
return  $v$ 
```

CS 3343 Analysis of Algorithms

Chapter 6: Slide - 20

## Fast Exponentiation

We can evaluate  $x^n$  quickly based on the binary expansion of the exponent. For example,

- $2^{10}$  can be evaluated by  $2^{10} = 2^{1010_2} = 2^8 * 2^2$ .
- $2^{13}$  can be evaluated by  $2^{13} = 2^{1101_2} = 2^8 * 2^4 * 2$ .
- Algorithm on next page corresponds to book's right to left algorithm.

CS 3343 Analysis of Algorithms

Chapter 6: Slide - 21

## Fast Exponentiation Algorithm

```
algorithm FastExpt( $x, n$ )  
  // Evaluates  $x^n$   
  // Input: A value  $x$  and an integer  $n \geq 0$   
  // Output: The value of  $x^n$   
   $term \leftarrow x$   
   $product \leftarrow 1$   
  while  $n > 0$  do  
    if  $n \bmod 2 = 1$  then  
       $product \leftarrow product * term$   
     $term \leftarrow term * term$   
     $n \leftarrow \lfloor n/2 \rfloor$   
  return  $product$ 
```