CS483-10 Elementary Graph Algorithms & Transform-and-Conquer

Instructor: Fei Li

Room 443 ST II

Office hours: Tue. & Thur. 1:30pm - 2:30pm or by appointments

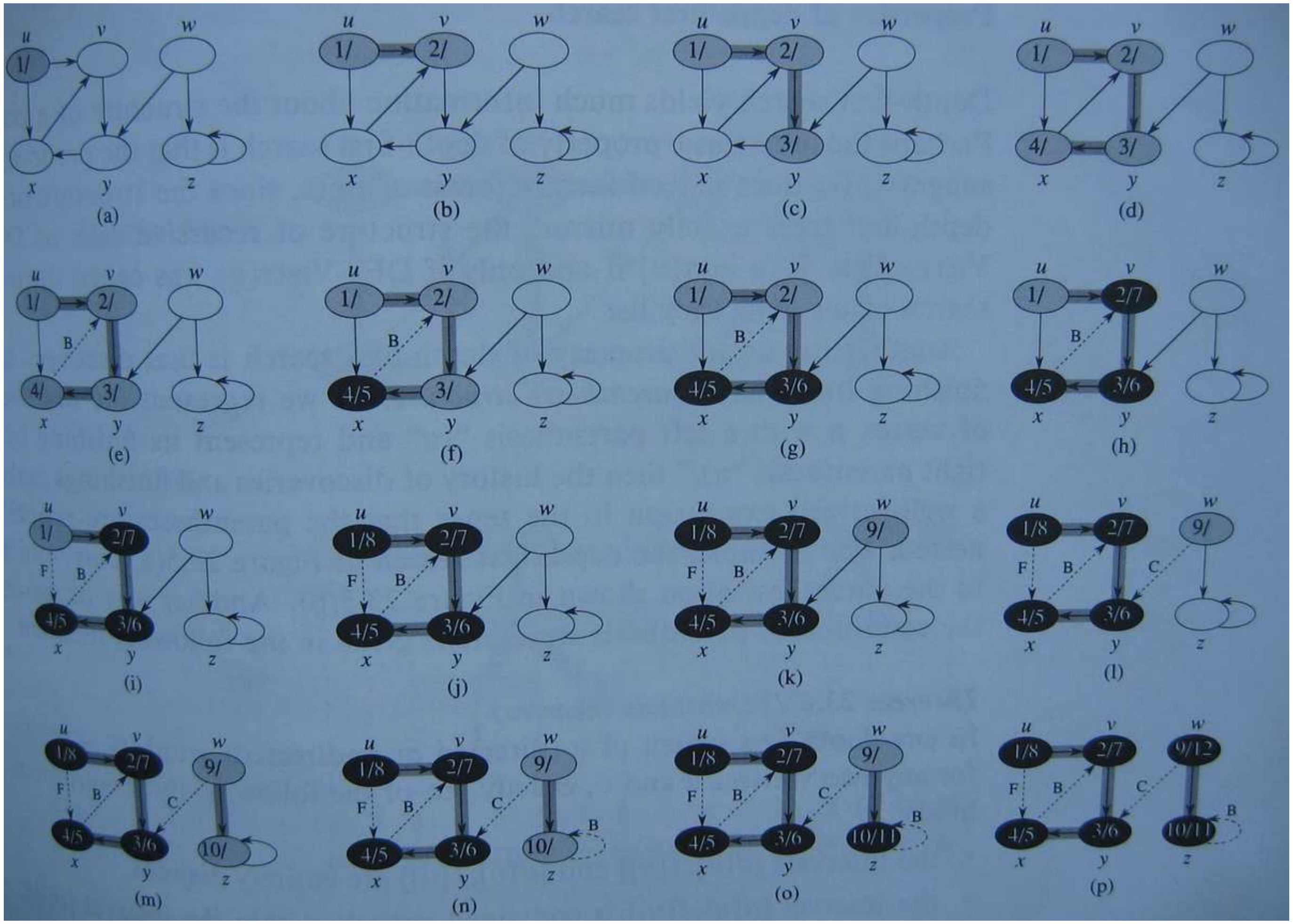
`lifei@cs.gmu.edu` with subject: CS483

`http://www.cs.gmu.edu/~ lifei/teaching/cs483_fall07/`

Based on *Introduction to the Design and Analysis of Algorithms* by Anany Levitin, Jyh-Ming Lien's notes, and *Introduction to Algorithms* by CLRS.

Outline

- ➢ Depth-first Search – cont
- ➢ Topological Sort
- ➢ Transform-and-Conquer – Gaussian Elimination


u v w
x y z
1/
(a)
1/ 2/
(b)
1/ 2/ 3/
(c)
1/ 2/ 4/ 3/
(d)
1/ 2/ B 4/ 3/
(e)
1/ 2/ B 4/5 3/
(f)
1/ 2/ B 4/5 3/6
(g)
1/ 2/7 B 4/5 3/6
(h)
1/ 2/7 F B 4/5 3/6
(i)
1/8 2/7 F B 4/5 3/6
(j)
1/8 2/7 9/ F B 4/5 3/6
(k)
1/8 2/7 9/ F B C 4/5 3/6
(l)
1/8 2/7 9/ F B C 4/5 3/6 10/
(m)
1/8 2/7 9/ F B C 4/5 3/6 10/ B
(n)
1/8 2/7 9/ F B C 4/5 3/6 10/11 B
(o)
1/8 2/7 9/12 F B C 4/5 3/6 10/11 B
(p)

Depth-first Search (DFS)

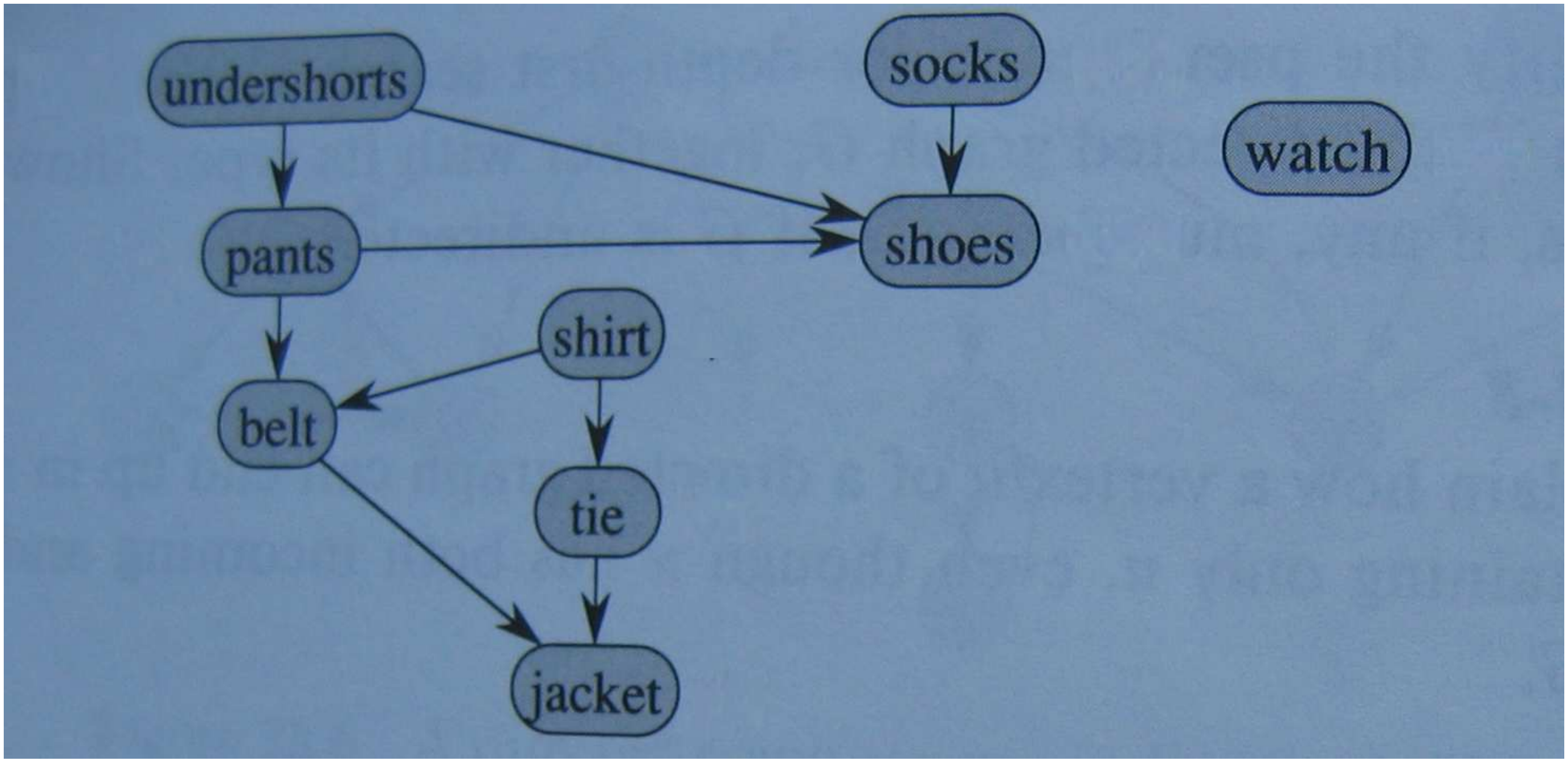
- The correctness proof: Use an induction method
- The overall running time of DFS is $O(|V| + |E|)$.
  - The time initializing each vertex is $O(|V|)$.
  - Each edge $(u, v) \in E$ is examined twice, once exploring u and once exploring v. Therefore takes $O(|E|)$ time.

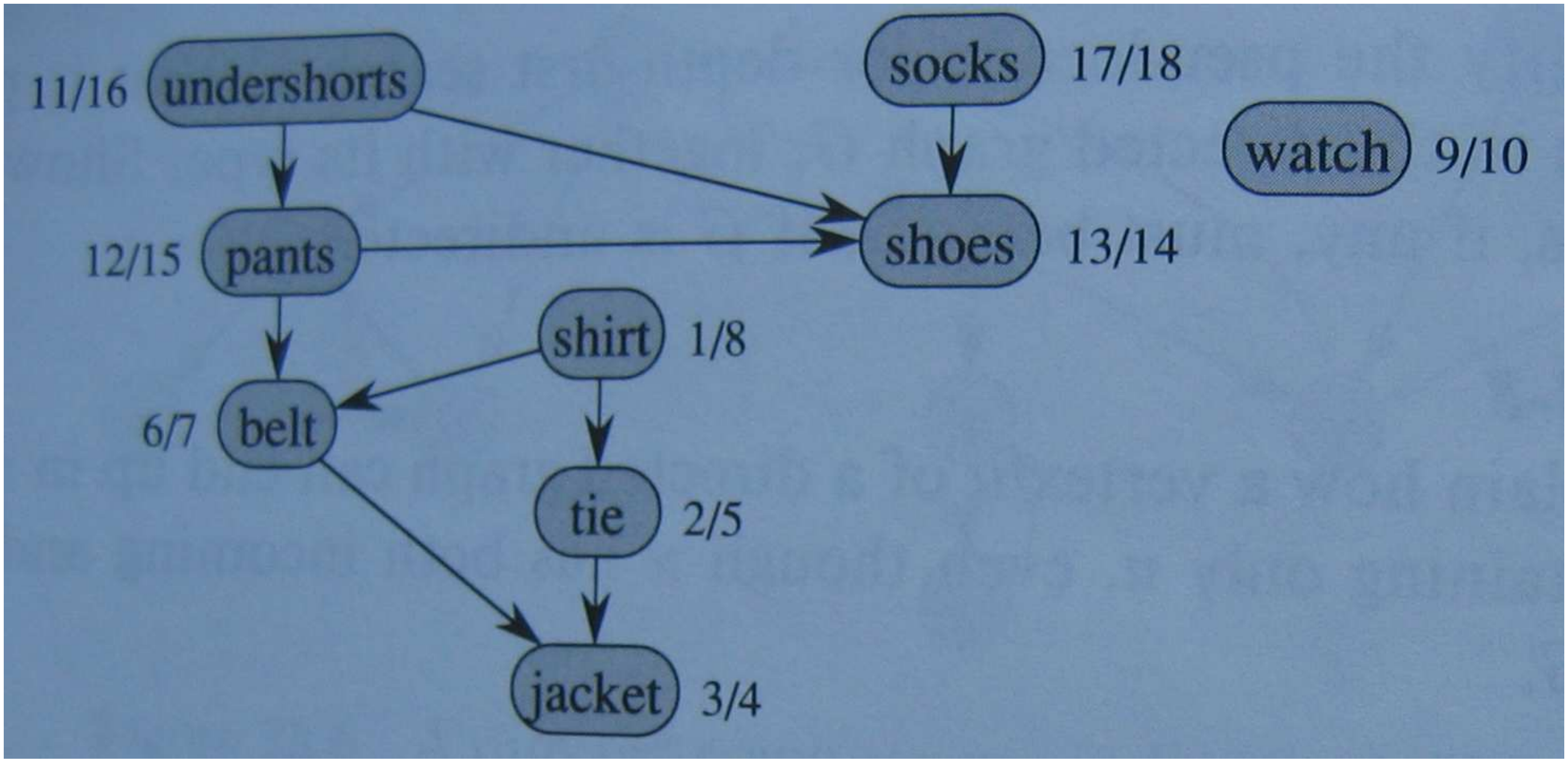
Outline

- Depth-first Search – cont
- Topological Sort
- Transform-and-Conquer – Gaussian Elimination

Topological Sort

- An application of DFS
- Input: a directed acyclic graph (DAG)
- Output: A linear ordering of all its vertices, such that if G contains an edge (u, v), then, u appears before v in the ordering.


undershorts
socks
watch
pants
shoes
shirt
belt
tie
jacket



11/16
undershorts
socks
17/18
watch
9/10
12/15
pants
shoes
13/14
shirt
1/8
6/7
belt
tie
2/5
jacket
3/4

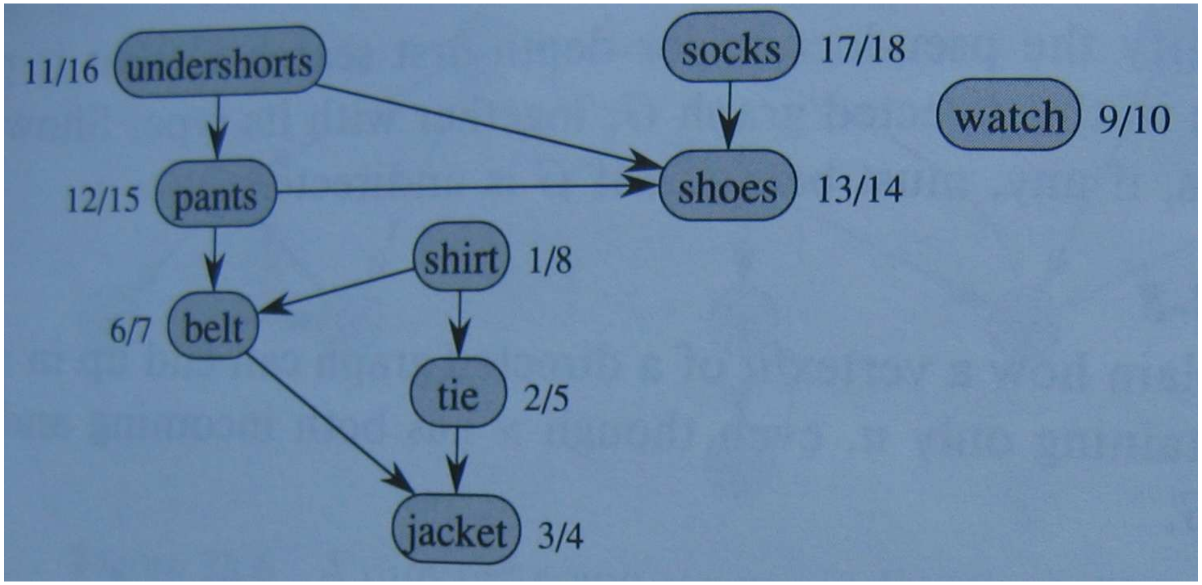
Topological-Sort

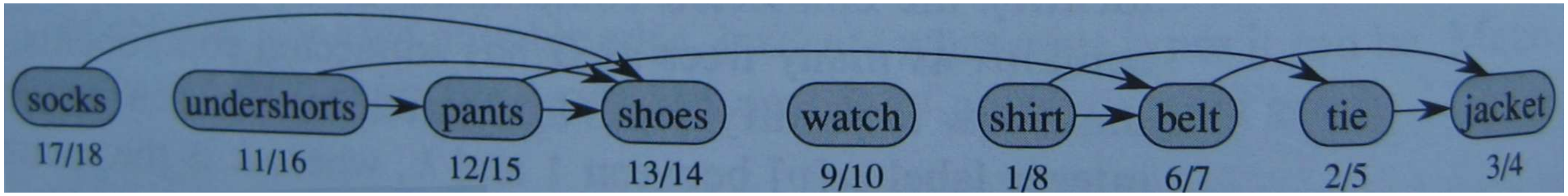
Algorithm 0.1: TOPOLOGICAL-SORT($G(V, E)$)

Call DFS(G) to compute finishing times $f[v]$ for each vertex v

As each vertex is finished, insert it onto the front of a linked list

return (the linked list of vertices)


11/16
undershorts
socks
17/18
watch
9/10
12/15
pants
shoes
13/14
shirt
1/8
6/7
belt
tie
2/5
jacket
3/4


socks
17/18
undershorts
11/16
pants
12/15
shoes
13/14
watch
9/10
shirt
1/8
belt
6/7
tie
2/5
jacket
3/4

Outline

- Depth-first Search – cont
- Topological Sort
- Transform-and-Conquer – Gaussian Elimination

Transform-and-Conquer

A problem is solved by a transformation

- To a simpler/more convenient instance of the same problem (**instance simplification**)

  - Ex: transforming unsorted to sorted

- To a different representation of the same instance (**representation change**)

  - Ex: transforming list to tree, tree to balanced tree, ... , etc.

- To a different problem for which an algorithm is already available (**problem reduction**)

  - Ex: transforming multiplication to addition

Transform-and-Conquer

- **Instance Simplification**
  - Element uniqueness
  - Mode (the most repeated element) of an array
  - Searching for a value in an array
- **Representation Change**
  - Gaussian elimination
  - AVL tree, 2-3 tree
  - Heap and heapsort
- **Problem Reduction**
  - Least common multiple
  - Paths in a graph - Linear programming (Chapter 10)

Instance Simplification

- Find if a given array contains unique elements.

  - What is the transform?

- Find the most repeated element.

  - What is the transform?

- Search for a value in an array (including binary search).

  - What is the transform?

- Quicksort

  - What is the transform?

Instance Simplification

- Find if a given array contains unique elements.

  - What is the transform? **Sorting**

- Find the most repeated element.

  - What is the transform? **Sorting**

- Search for a value in an array (including binary search).

  - What is the transform? **Sorting**

- Quicksort

  - What is the transform? **Randomization**

Instance Simplification - Presorting

Instance Simplification: Solve a problem's instance by transforming it into another simpler/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 directed Acyclic graphs (DAGs).
- Presorting is used in many geometric algorithms.

How Fast Can We Sort?

- Efficiency of algorithms involving sorting depends on efficiency of sorting.
- Theorem (see Sec. 11.2): $\lceil \log_2 n! \rceil \approx n \log_2 n$ comparisons are necessary in the worst case to sort a list of size n by any comparison-based algorithm.
- Note: About $n \log_2 n$ comparisons are also sufficient to sort array of size n (by mergesort).

Searching with Presorting

- Problem: Search for a given K in $A[1...n]$
- Presorting-based algorithm:
  1. Sort the array by an efficient sorting algorithm
  2. Apply binary search
- Efficiency: $\Theta(n \log_2 n) + O(\log_2 n) = \Theta(n \log_2 n)$
- Good or bad?
- Why do we have our dictionaries, telephone directories, etc. sorted?

Searching with Presorting

- Problem: Search for a given K in $A[1...n]$
- Presorting-based algorithm:
  1. Sort the array by an efficient sorting algorithm
  2. Apply binary search
- Efficiency: $\Theta(n \log_2 n) + O(\log_2 n) = \Theta(n \log_2 n)$
- Good or bad?
- Why do we have our dictionaries, telephone directories, etc. sorted?

  Cumulative cost is reduced.

  $M \cdot n$ vs. $n \log_2 n + M \cdot \log_2 n$, given M is large.

Element Uniqueness with Presorting

- Brute force algorithm
  - Compare all pairs of elements
  - Efficiency: $O(n^2)$
- Presorting-based algorithm
  1. Sort by efficient sorting algorithm (e.g. mergesort)
  2. Scan array to check pairs of adjacent elements

  Efficiency: $\Theta(n \log_2 n) + O(n) = \Theta(n \log_2 n)$

Transform-and-Conquer

- **Instance Simplification**
  - Element uniqueness
  - Mode (the most repeated element) of an array
  - Searching for a value in an array
- **Representation Change**
  - Gaussian elimination
  - AVL tree, 2-3 tree
  - Heap and heapsort
- **Problem Reduction** (Chapter 10)
  - Least common multiple
  - Paths in a graph - Linear programming

Representation Change: Gaussian Elimination

Problem: Solve the linear system of a set of n linear equations and n variables.

➢ Example 1:

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

Representation Change: Gaussian Elimination

Problem: Solve the linear system of a set of n linear equations and n variables.

- Example 1:

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

- Example 2:

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

Representation Change: 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.
- **Base**: If we add a multiple of one equation to another, the overall system of equations remains equivalent.

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

Transformation by Gaussian Elimination

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

- for $f \leftarrow 1$ 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.

Gaussian Elimination: Example

$$\begin{aligned} 2x_1 - 4x_2 + x_3 &= 6 \\ 3x_1 - x_2 + x_3 &= 11 \\ x_1 + x_2 - x_3 &= -3 \end{aligned}$$

$$\begin{array}{cccc} 2 & -4 & 1 & 6 \\ 3 & -1 & 1 & 11 \\ 1 & 1 & -1 & -3 \end{array}$$

Gaussian Elimination: Example

$$\begin{aligned} 2x_1 - 4x_2 + x_3 &= 6 \\ 3x_1 - x_2 + x_3 &= 11 \\ x_1 + x_2 - x_3 &= -3 \end{aligned}$$

$$\begin{array}{rrrrl} 2 & -4 & 1 & 6 & \\ 3 & -1 & 1 & 11 & \text{row } 2 - \frac{3}{2} \times \text{row } 1 \\ 1 & 1 & -1 & -3 & \text{row } 3 - \frac{1}{2} \times \text{row } 1 \end{array}$$

$$\begin{array}{rrrr} 2 & -4 & 1 & 6 \\ 0 & 5 & -\frac{1}{2} & 2 \\ 0 & 3 & -\frac{3}{2} & -6 \end{array}$$

Gaussian Elimination: Example

$$\begin{aligned} 2x_1 - 4x_2 + x_3 &= 6 \\ 3x_1 - x_2 + x_3 &= 11 \\ x_1 + x_2 - x_3 &= -3 \end{aligned}$$

$$\begin{array}{ccccl} 2 & -4 & 1 & 6 & \\ 3 & -1 & 1 & 11 & \text{row } 2 - \frac{3}{2} \times \text{row } 1 \\ 1 & 1 & -1 & -3 & \text{row } 3 - \frac{1}{2} \times \text{row } 1 \end{array}$$

$$\begin{array}{ccccl} 2 & -4 & 1 & 6 & \\ 0 & 5 & -\frac{1}{2} & 2 & \\ 0 & 3 & -\frac{3}{2} & -6 & \text{row } 3 - \frac{3}{5} \times \text{row } 2 \end{array}$$

$$\begin{array}{cccc} 2 & -4 & 1 & 6 \\ 0 & 5 & -\frac{1}{2} & 2 \\ 0 & 0 & -\frac{6}{5} & -\frac{36}{5} \end{array}$$

Gaussian Elimination: Example

- We have:

$$\begin{aligned} 2x_1 - 4x_2 + x_3 &= 6 \\ 5x_2 - \frac{1}{2}x_3 &= 2 \\ -\frac{6}{5}x_3 &= -\frac{36}{5} \end{aligned}$$

- Then we can solve x_3, x_2, x_1 by backward substitution:

$$\begin{aligned} x3 &= (-\frac{36}{5})/(-\frac{6}{5}) = 6 \\ x2 &= (2 + (\frac{1}{2}) \times 6)/5 = 1 \\ x1 &= (66 + 4 \times 1)/2 = 2 \end{aligned}$$

Gaussian Elimination Algorithm

Algorithm 0.2: GE($A[1 \cdots n, 1 \cdots n], b[1 \cdots n]$)

Append b to A as the last column

for $i \in \{1 \cdots n-1\}$

do $\begin{cases} \textbf{for } j \in \{i+1 \cdots n\} \\ \textbf{do } \begin{cases} \textbf{for } k \in \{j \cdots n\} \\ \textbf{do for } A[j,k] = A[j,k] - \frac{A[i,k]A[j,i]}{A[i,i]} \end{cases} \end{cases}$

Backward Substitution Algorithm

Algorithm 0.3: BS($A[1 \cdots n, 1 \cdots n+1]$)

for $j \leftarrow \{n \cdots 1\}$

do $\begin{cases} t \leftarrow 0 \\ \textbf{for } k \leftarrow \{j+1 \cdots n\} \\ \quad \textbf{do } t \leftarrow t + A[j,k] \times x[k] \\ x[j] \leftarrow (A[j, n+1] - t)/A[j,j] \end{cases}$

Solve $Ax = b$ using Gaussian Elimination

- To solve a linear system $Ax = b$: We will call GE(A,b) and then x = BS(A)
- Time Complexity:

$$T(n) = n^2 + (n-1)^2 + \cdots + (n-(n-1))^2 =$$
$$1^2 + 2^2 + \cdots + n^2 \approx \frac{1}{3}n^3 = \Theta(n^3).$$

More about Gaussian Elimination

- Issues with Gaussian Elimination
  - The value of $A[j,i]/A[i,i]$ is repetitively computed
  - Small $A[i,i]$ make the algorithm unstable (numerical errors), e.g., $A[j,i]/A[i,i]$ will be too large to cause over flow.
- Solution: pivoting: always select the largest $A[i,i]$

Better Gaussian Elimination Algorithm

Algorithm 0.4: GE($A[1 \cdots n, 1 \cdots n], b[1 \cdots n]$)

Append b to A as the last column

for $i \in \{1 \cdots n-1\}$

do
- **for** $j \in \{i+1 \cdots n\}$
  - **do if** $|A[j,i]| > |A[pivot,i]|$
  - **then** $pivot \leftarrow j$
- **for** $j \in \{i \cdots n+1\}$
  - **do** swap($A[i,k], A[pivot,k]$)
- **for** $j \in \{i+1 \cdots n+1\}$
  - **do**
    - $temp \leftarrow \frac{A[j,i]}{A[i,i]}$
    - **for** $k \in \{j \cdots n\}$
      - **do for** $A[j,k] = A[j,k] - A[i,k] \times temp$

Why Gaussian Elimination?

- Solve linear equations $Ax = b$
- LU decomposition
- Matrix inverse
- Compute Determinant

LU Decomposition

- **LU decomposition ($A = LU$)**

  Decompose a matrix into two matrices: an upper triangular matrix and a lower triangular matrix

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

- Gaussian elimination can be used to compute L (or U), which is then used to compute U (or L).

- LU decomposition is good if you have different bs, i.e.,
  $Ax = b \Rightarrow L(Ux) = b \Rightarrow Ux = b'$

Matrix Inverse

- Compute Inversion

  A^{-1} of an invertible $n \times n$ matrix A. Recall that $AA^{-1} = I$

- We can use Gaussian elimination (Gauss-Jordan elimination to be precise) to compute inverse of a matrix.

- Not all $n \times n$ matrices can be invertible. Such matrices are called singular.

Matrix Inverse

Example:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} A^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

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

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

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} A^{-1} = \begin{bmatrix} -2 & 1 \\ 3/2 & -1/2 \end{bmatrix}$$

Matrix Determinant

- The determinant of a matrix A is denoted $\det A$ or $|A|$
- When $\det A \neq 0$, A is invertible.
- Example:

$$\det \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = aei - afh - bdi + bfg + cdh - ceg$$

- Time complexity of a brute force algorithm? $O(n!)$

Matrix Determinant (cont.)

➢ Using Gaussian elimination, and some properties of determinant:

- Interchanging two rows changes the sign of the determinant.
- Multiplying a row by a scalar multiplies the determinant by that scalar.
- Replacing any row by the sum of that row and any other row does **NOT** change the determinant.
- The determinant of a triangular matrix (upper or lower triangular) is the product of the diagonal elements.

➢ Example:

$$A = \begin{bmatrix} 5 & 3 \\ -4 & -2 \end{bmatrix} = 5 \cdot \begin{bmatrix} 1 & 0.6 \\ -4 & -2 \end{bmatrix} = 5 \begin{bmatrix} 1 & 0.6 \\ 0 & 0.4 \end{bmatrix}$$

$D = x$ and $x/5 = 0.4 \Rightarrow x = 2$