CSCE350: Data Structures and Algorithms

Spring 2022

Dr. Yan Tong

Health and Safety

Follow current COVID-19 guidelines
https://sc.edu/safety/coronavirus/safety_guidelines

Face coverings

Seat map – keep a track who sit together for contact tracing

Course Information

Instructor: Dr. Yan Tong

Email: tongy@cse.sc.edu

Office: Storey Innovation Center 2273

Office Hours: By appointment

About Me

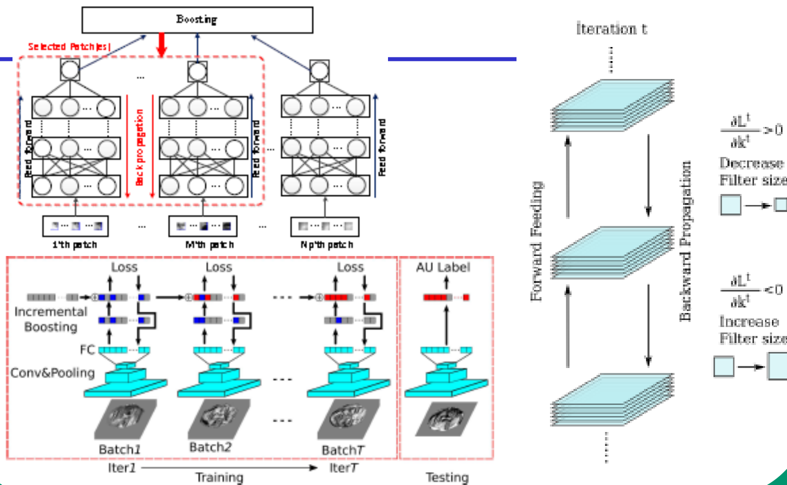
Associate Professor, UofSC	Jan. 2017 – present
Acting Graduate Director, CSE	Jan. 2019 – June 2019
Assistant Professor, UofSC	Dec. 2010 – Dec. 2016
Research Scientist, GE GRC	Jan. 2008 – Nov. 2010
Ph. D. from RPI	Dec. 2007

Selected Honors and Awards

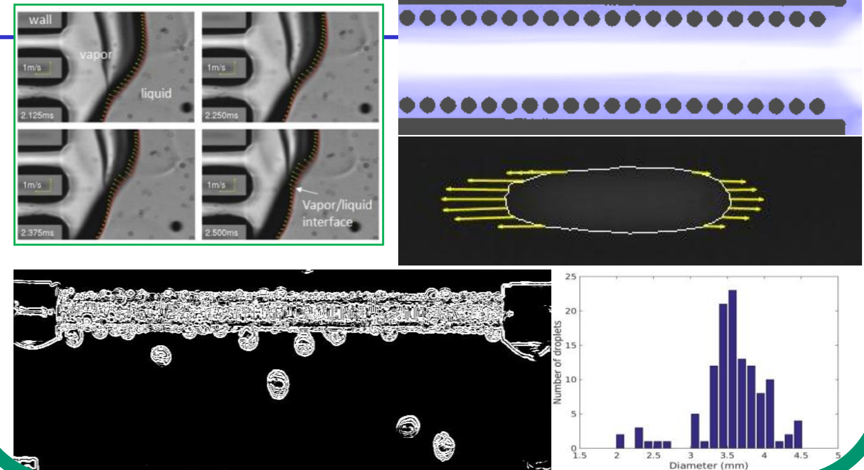
- Univ. of South Carolina Breakthrough Star, 2014
- NSF CAREER Award, 2012
- GE Bronze Patent Award, 2009
- GE Level 3 Award (the second highest level), 2009

Dr. Tong’s Main Research Areas

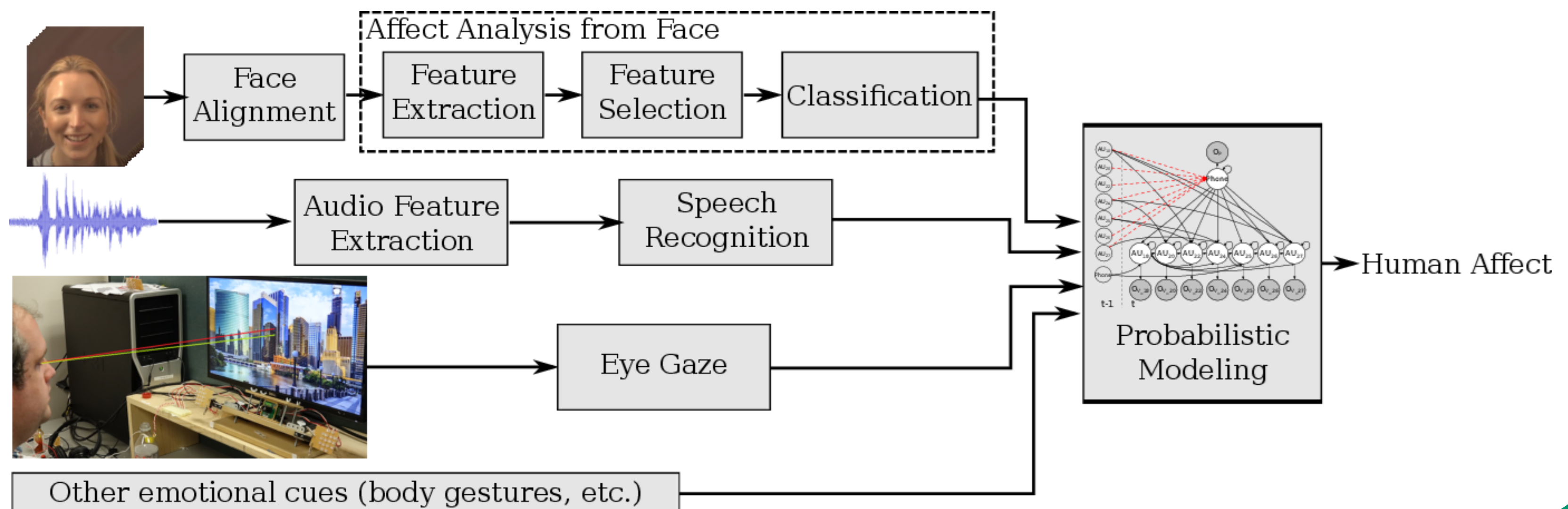
Fundamental Research in CV/ML



CV/ML Enabled Data Analysis



Multimodal Information Fusion



Today’s Agenda

Welcome.

Various administrative issues.

What is algorithm?

What is this course about?

Class Communication

Class homepage

http://www.cse.sc.edu/~tongy/csce350/csce350.html

Course syllabus

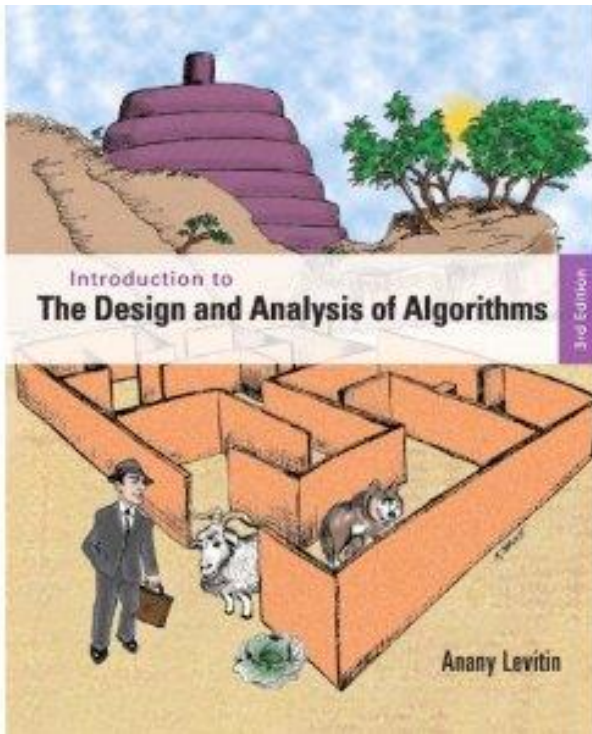
Blackboard (blackboard.sc.edu)

- Course syllabus
- Lecture notes
- Discussion board
- Solutions to homework assignments, quizzes, and exams
- **Submission of homework/programming assignments, quizzes, and exams**

Check them regularly for

- Homework assignments and solutions, lectures, and more
- important announcements related to this course
- some useful links and additional readings

Required Textbook



Introduction to the Design and Analysis of Algorithms, 3rd Edition

Anany V. Levitin, Villanova University

Addison Wesley

Grading Policy

- A (90-100%)
- B+ (86-89%)
- B (80-85%)
- C+ (76-79%)
- C (70-75%)
- D+ (66-69%)
- D (60-65%)
- F (0-59%)

Your scores of homework, programming assignments, quizzes, exams, etc. will be available to you at Blackboard when graded.

Your Grade Consists of

Midterm exams (2) (15% each)

Final exam (20%)

Homework assignments (6) (4% each)

Programming assignments (4) (21%, different weights applied)

Quizzes (5) (1% each)

Exams

All exams are closed-book and closed-notes, except

- a single-side letter-size cheat sheet for each midterm
- a double-side letter-size cheat sheet for the final exam.

Midterms: during class time. The dates and materials covered in the midterms will be announced later

Final exam: May 3, Tuesday, 9:00 – 11:30 a.m.

Quizzes

Quizzes will be held in class and announced the lecture before.

Quizzes are closed-book, but open-notes either printed or handwritten.

Submission of Exams and Quizzes

Exams and quizzes will be conducted through **Blackboard**.

- You may need to scan and upload your answers

Hard copies of quizzes or exams are available upon request. **Please let me know if you prefer a hard copy.**

Please make sure you bring a laptop during the exam/quiz days.

About Homework Assignments

- Must be completed independently by yourself while peer discussion is encouraged
- A due date will be accompanied with each homework. The due time will be at the **beginning of the class**.
- Print or handwritten
- Submit via Blackboard **before class starts**
- Without the special permission from the instructor, **NO late** homework will be accepted – Refer to late submission policy

Programming Assignment

For each programming assignment,

- While peer discussion is encouraged, **you must complete it independently by yourself** unless a teamwork is permitted by the instructor
- **You'll be asked to implement a program to solve a problem**
- **Requirements**
  - Code written in **C or C++** (correct and clearly commented)
  - A script or readme file including the instructions to compile and run the code
  - Code will be tested on department linux machines
  - Code that does not compile will not be graded and get a **0** automatically
  - Code needs to be turned in through Blackboard
- A due date will be accompanied with each programming assignment. **Without the special permission from the instructor, NO late submission will be accepted** – Refer to late submission policy

Code of Student Academic Responsibility

Violations of the University's Honor Code include, but are not limited to improper citation of sources, using another student's work, and any other form of academic misrepresentation.

Any violation will result in a minimum academic penalty of **a grade of Zero** of the assignment and will result in additional disciplinary measures.

Violations of the University's Honor Code will be reported to the Office of Student Conduct and Academic Integrity.

Topics Covered

1. Structured programming, stacks, queues, lists (3 hours)
2. Determining the Running Time of Programs, Order of Magnitude Analysis (6 hours)
3. Brute force (3 hours)
4. Divide-and-Conquer (4 hours)
5. Dynamic Programming (6 hours)
6. Transform-and-Conquer (4 hours)
7. The Greedy Technique (3 hours)
8. Decrease-and-Conquer (3 hours)
9. Graphs (3 hours)
10. Reviews and exams (4 hours)
11. More as time permits (invited talks)

The time allocated for each topic is approximated.

What is an Algorithm?

In General: What is an Algorithm?

Recipe, process, method, technique, procedure, routine,… with following requirements:

- **Finiteness:** terminates after a finite number of steps
- **Definiteness:** each step is unambiguously specified
- **Input:** valid inputs are clearly specified
- **Output:** can be proved to produce the correct output given a valid input
- **Effectiveness:** steps are sufficiently simple and basic

Some Important Points

- Each step of an algorithm is unambiguous
- The range of inputs has to be specified carefully
- The same problem may be solved by different algorithms
- Different algorithms may take different time/space to solve the same problem – we may prefer one to the other
- The same algorithm can be implemented in different ways

Why Study Algorithms?

Theoretical importance

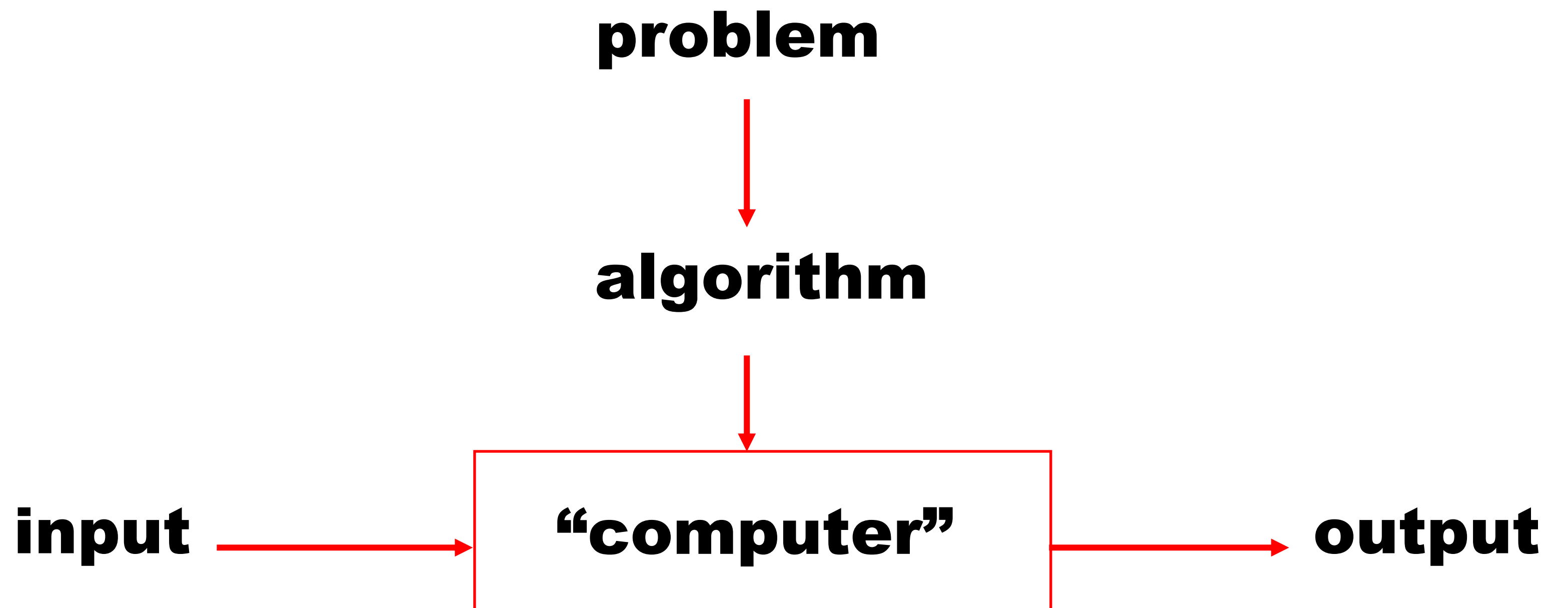
- the core of computer science

Practical importance

- A practitioner's toolkit of known algorithms
  - Expedia travel plan, yahoo driving directions, Google search, ..
- Framework for designing and analyzing algorithms for new problems

Definition of Algorithm

An ***algorithm*** **is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input, in a finite amount of time.**



Example: Finding Greatest Common Divisor gcd(*m*,*n*)

Input: *m* and *n* are two nonnegative integers, (at least one of them is non-zero), m>n (Note: the range of input is specified)

Output: gcd(*m*,*n*), the greatest common divisor, i.e., the largest integer that divides both *m* and *n*

Euclid's algorithm: Based on $gcd(m,n)=gcd(n, m \bmod n)$ and

$gcd(m,0)= m$

For example:

gcd(60, 24)

=gcd(24,12)

=gcd(12,0)

=12

Will this algorithm eventually come to a stop? Why?

Finding Greatest Common Divisor: Euclid algorithm

ALGORITHM $Euclid(m,n)$

while $n \neq 0$

$r \leftarrow m \bmod n$

$m \leftarrow n$

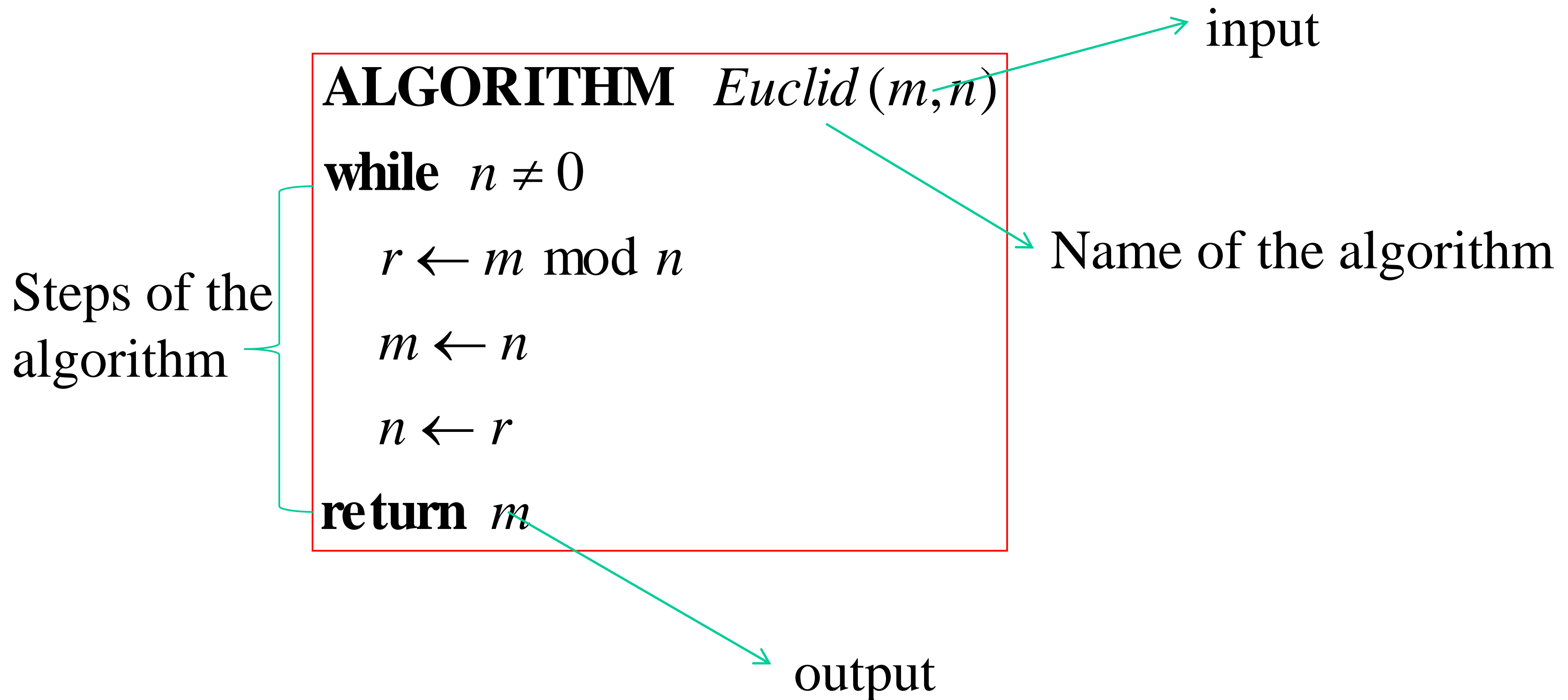
$n \leftarrow r$

return m

Pseudocode:

- Easy for human reading
- Independent of programming language
- Compact
- Following structural convention of programming language, e.g.
  - looping structure
  - If-else branching

Finding Greatest Common Divisor: Euclid algorithm



In this course, you are required to write algorithms in pseudocode instead of the real code in some special language.

Another Algorithm for Finding gcd(*m*,*n*)

Note that $0 < \gcd(m,n) \le \min(m,n)$ **, the pseudocode is**

$$
\begin{aligned}
&\textbf{ALGORITHM}\ \gcd(m,n) \\
&t \leftarrow \min(m,n) \\
&\textbf{while}\ (m \bmod t \neq 0)\ or\ (n \bmod t \neq 0) \\
&\quad t \leftarrow t-1 \\
&\textbf{return}\ t
\end{aligned}
$$

The algorithm is derived based on the definition of **gcd**.

The Third Algorithm for Finding gcd(*m*,*n*)

Based on the method you learned in middle school

- Step 1: Find the prime factors of *m*
- Step 2: Find the prime factors of *n*
- Step 3: Identify all the common factors
- Step 4: Compute the product of these identified common factors as gcd(*m*,*n*)

Example:

$$60 = 2 \cdot 2 \cdot 3 \cdot 5 \qquad 24 = 2 \cdot 2 \cdot 2 \cdot 3$$

$$\Rightarrow \gcd(60,24) = 2 \cdot 2 \cdot 3 = 12$$

Problems: How to find all the prime factors of an integer?

(Sieve algorithm: See page 6-7 of the textbook)

Comparison

All the three algorithms find the *gcd*, which one is the best?

Euclid's Algorithm: $5 \log_{10} n$ (*worst case*)
Algorithm 2: n (worst case)
Algorithm 3: $c * n\log\log n$

Fundamentals of Algorithmic Problem Solving

Understanding the problem

Ascertaining the capabilities of a computational device

- Random-access machine (RAM) → sequential algorithms
- Speed and space

Choosing between exact and approximate problem solving

Deciding on appropriate data structure

- Array, linked list, tree, etc.

Algorithm design techniques

Methods of specifying an algorithm

- Pseudocode (for, if, while, //, ←, indentation…)

Prove an algorithm's correctness – mathematic induction

Analyzing an algorithm – Simplicity, efficiency, generality

Coding an algorithm

In general

A good algorithm is a result of repeated effort and rework

- Better data structure
- Better algorithm design
- Better time and/or space efficiency
- Easy to implement

Some Well-known Computational Problems

Sorting

Searching

Shortest paths in a graph: e.g., airline planning

Minimum spanning tree: e.g., planning for laying cable

Primality testing

Traveling salesman problem: e.g., soldering in manufacturing microchips

Knapsack problem: e.g, finding the least wasteful way to cut raw materials, selection of investments

Example: Sorting

Statement of problem:

- *Input:* A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$
- *Output:* A reordering of the input sequence $\langle a'_1, a'_2, \ldots, a'_n \rangle$
- so that $a'_i \leq a'_j$ whenever $i < j$

Instance: The sequence $\langle 5, 3, 2, 8, 3 \rangle$

Algorithms:

- Selection sort
- Insertion sort
- Merge sort
- (many others)

Two desired properties:

- stable (preserving order for equal elements) and
- in place (no extra memory)

Selection Sort

Input: array $a[1], a[2], \ldots, a[n]$

Output: array $a[1..n]$ **sorted in non-decreasing order**

Algorithm:

for $i = 1$ to n
    swap $\boldsymbol{a}[i]$ with smallest of $a[i], \ldots, a[n]$

Selection Sort

```
ALGORITHM  SelectionSort(A[0..n − 1])
    //Sorts a given array by selection sort
    //Input: An array A[0..n − 1] of orderable elements
    //Output: Array A[0..n − 1] sorted in ascending order
    for i ← 0 to n − 2 do
        min ← i
        for j ← i + 1 to n − 1 do
            if A[j] < A[min]   min ← j
        swap A[i] and A[min]
```

This Course is Focused on

How to design algorithms

How to describe algorithms -- pseudocode

How to prove correctness

How to analyze algorithms -- efficiency

- Theoretical analysis
- Empirical analysis

Analysis of Algorithms

How good is the algorithm?

- Correctness
- Time efficiency
- Space efficiency

Does there exist a better algorithm?

- Lower bounds
- Optimality

Algorithm Design Strategies

Brute force

Decrease & conquer

Divide & conquer

Transform & conquer

Greedy approach

Dynamic programming

Backtracking and branch & bound

Space and time tradeoffs: e.g., hashing

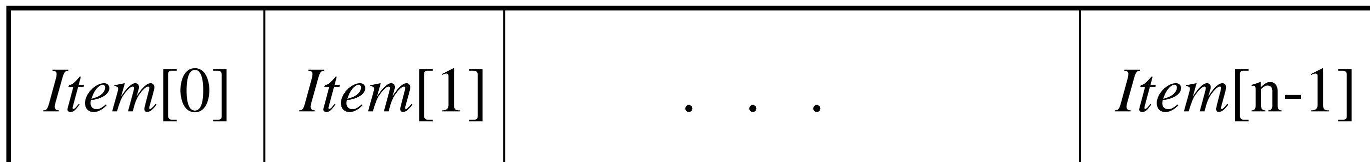
Fundamental Data Structure

Data Structure: a particular scheme of organizing related data items

We know you are familiar with most important ones: array, list, graph, …

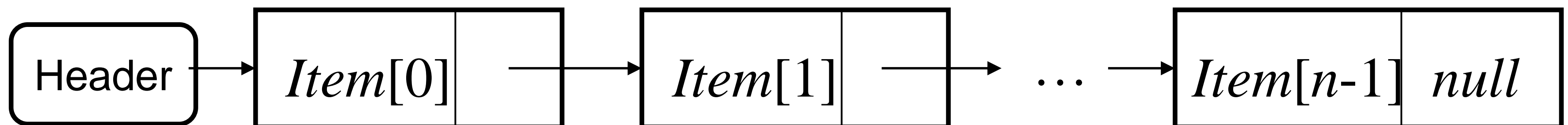
Array

- **A sequence of *n* items of <u>the same data type</u>**
- **Store <u>contiguously</u> in computer memory**
- **Can be accessed by its index**
  - Index is an integer ranged from [0..*n*-1], [1..*n*], or [*low*, *high*]
- **Each and every element of an array can be <u>accessed in the same constant amount of time</u>**
- **Can be used to implement string, a sequence of symbols**



Linked List

- A sequence of elements called nodes
- Each node contains data and pointers
- Single linked list – single pointer
- To access a particular node, we start from header and traverse the list
- The time needed for access a different node is different
- But it does not require reservation of the memory
- Easy to do insertions and deletions



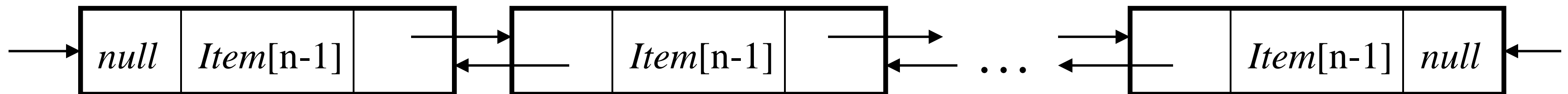
Doubly Linked List

➢ **Every node, except the first and the last, contains pointers to both its successor and its predecessor**

➢ **Why we need doubly linked list?**

- Single linked list can only go forward but cannot go backward

➢ **How to do deletions and insertions in doubly linked list?**



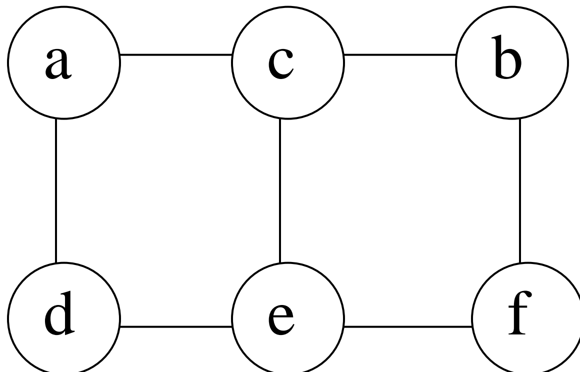
Special Linked List

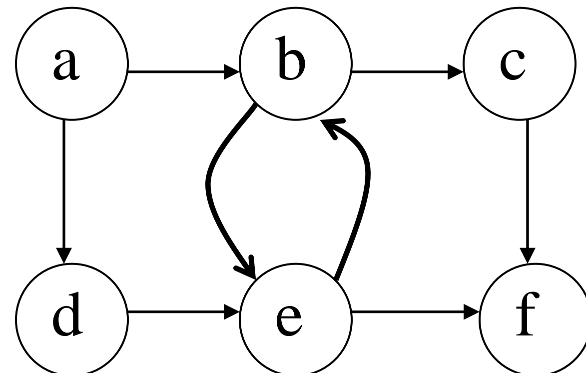
- **Stack – “last in first out” (LIFO)**
  - Push
  - Pop
  - Top
- **Queue – “first in first out” (FIFO)**
  - Enqueue
  - dequeue
- **Priority queue**
  - Heap (finding/deleting the largest, insertion)

Graph

- **Undirected and directed (digraph):** ***G*=<*V*,*E*>**
  - vertices (V) and edges (E)
  - number of vertices (|V|) and number of edges (|E|)
  - Undirected graph
    - Endpoints: e.g., two endpoints of an edge (a,b)
  - Directed graph/Digraph
    - Tail and head: e.g., tail (a) and head (b) of an edge (a,b)

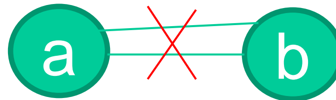
Write down the *V* and *E* for the following two graphs





Graph

- **Loop:** edge connecting a vertex to itself, e.g. a—a
- No multiple edges between the same pair of vertices in an **undirected graph**



- **Complete graph:** an undirected graph, where every pair of vertices is connected by an edge, denoted as $K_{|V|}$
- **Number of edges in an undirected graph without loops**

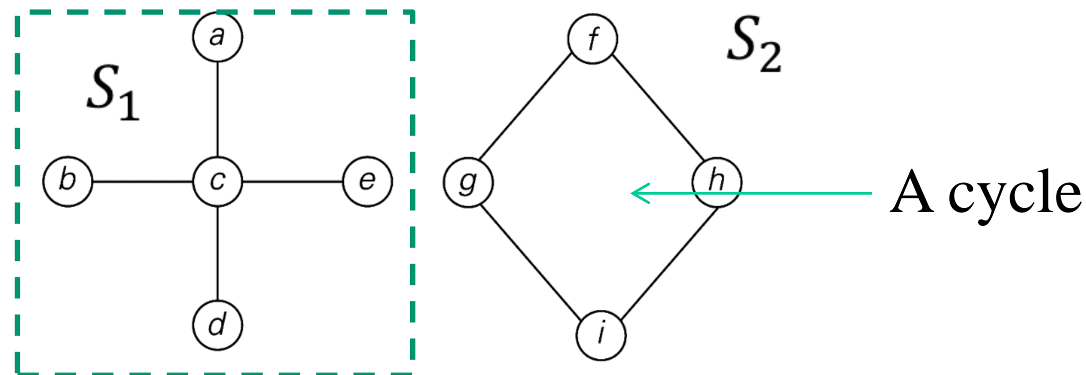
$$0 \le |E| \le |V|(|V|-1)/2$$

complete graph

- **Dense graph ←→ sparse graph**

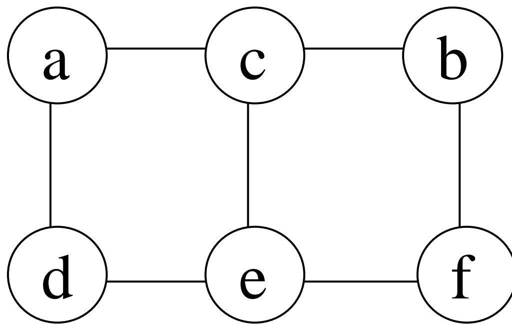
Graph

- **Path: a sequence of adjacent (connected by an edge) vertices starting from *u* and ending at *v***
  - simple path: all traversed edges/vertices are distinct
  - Length of the path -- # of traversed edges
- **Connected graph – there exists a path between each pair of vertices**
- **Cycle – a simple path between the same vertex**
- **Acyclic graph – a graph with no cycles**



Graph Representation

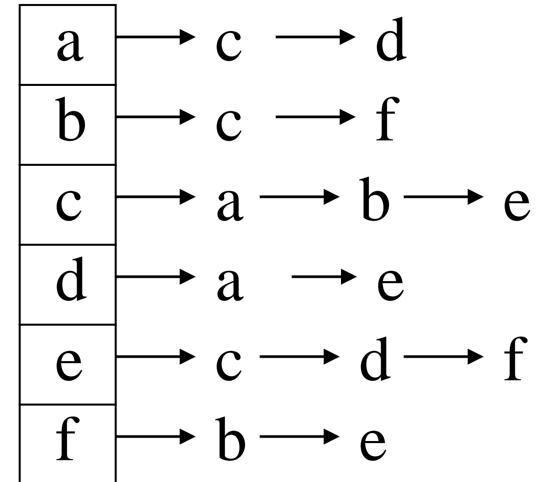
Adjacency matrix and adjacency linked list



Adjacency matrix

$$
\begin{bmatrix}
0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 0
\end{bmatrix}
$$

Adjacency linked list



When should we use adjacency matrix and when adjacent linked list?

Graph sparseness