

CMP329/IFT303: Data Structures and Algorithms Analysis (3 units)

Lecture 1: Introduction

MR. M. YUSUF

Course Outline

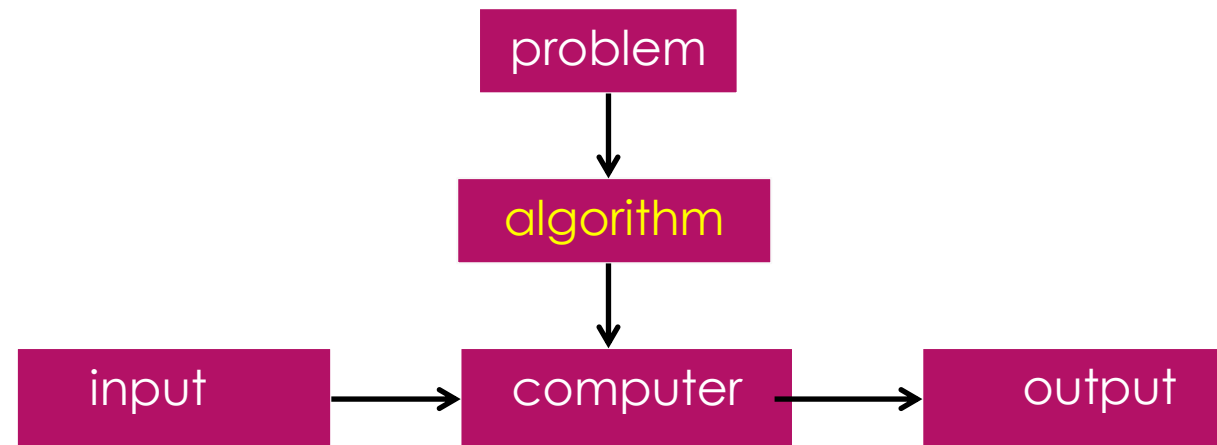
- ▶ Introduction and review
- ▶ Introduction to Arrays, Lists, stacks, and queues, Trees, Graphs
- ▶ Algorithm Efficiency Analysis Framework
- ▶ **Algorithm design Techniques**
- ▶ Sorting and Searching algorithms
- ▶ Tree Algorithm
- ▶ Graph Algorithms

Outline

- ▶ What is an Algorithm?
 - ▶ Properties of Algorithms
- ▶ Algorithm design and analysis process.
 - ▶ How to Understand program development problem
 - ▶ Algorithm Design Techniques
 - ▶ ...
- ▶ Important Problem Types
- ▶ Fundamental Data Structure

What is an Algorithm?

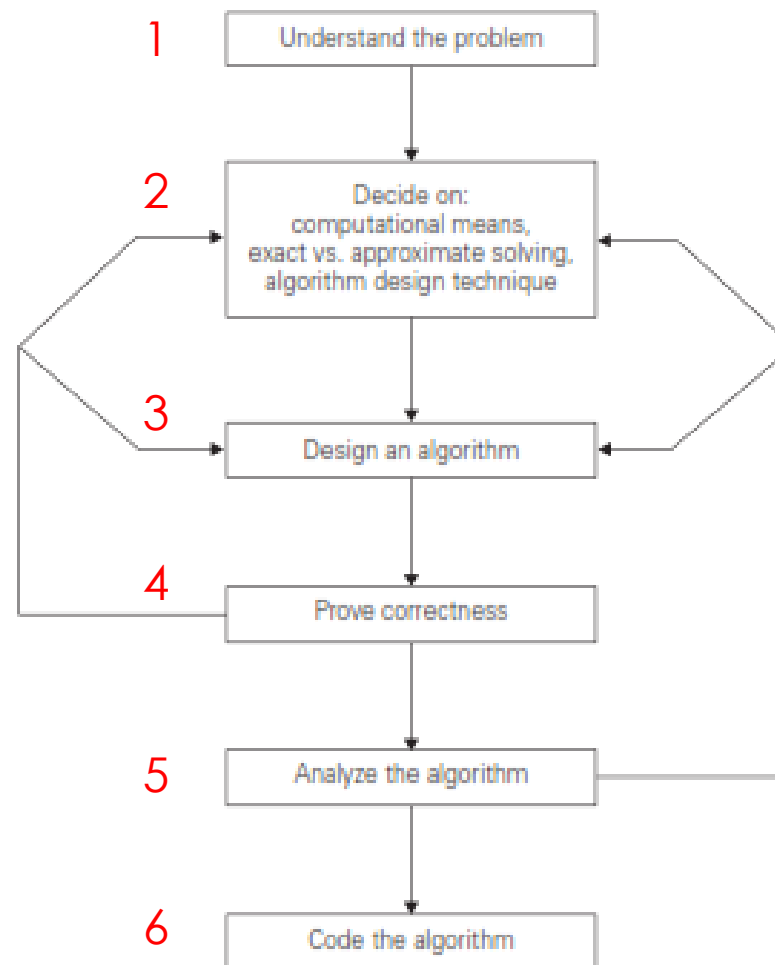
- ▶ A sequence of **unambiguous** instruction for solving a problem that is, for obtaining a required **output** for any legitimate **input** in a **finite** amount of time.



Properties of Algorithms

- ▶ **Finiteness:** terminates after a finite number of steps.
- ▶ **Definiteness:** each step must be rigorously and unambiguously specified, e.g., “stir until **lumpy**”
- ▶ **Input:** valid inputs must be clearly specified
- ▶ **Output:** data that result upon completion of the algorithm must be specified
- ▶ **Effectiveness:** steps must be simple and basic, e.g., **check if m is prime**

Algorithm Design and Analysis Process.



Algorithm Design and Analysis Process. Con't

1. Understanding the Problem

- ▶ Ask questions, do a few small examples by hand, think about special cases, etc.
- ▶ An input is an instance of the problem the algorithm solves
- ▶ Specify exactly the set of instances the algorithm needs to handle

2.1 Decide on Exact vs. approximate solution

- ▶ Approximate solution cannot solve exactly, e.g., extracting square roots, solving definite integrals, etc.
- ▶ Algorithms for exact solution can be unacceptably slow

Algorithm Design and Analysis Process. Con't

2.2 Algorithm Design Techniques

- ▶ is a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing such as.
- ▶ Brute force (Chapter 3)
- ▶ Decrease and Conquer (Chapter 4)
- ▶ Divide and Conquer (Chapter 5)
- ▶ Transform and Conquer (Chapter 6)
- ▶ Space and time tradeoffs (Chapter 7)
- ▶ Dynamic Programming (Chapter 8)

Algorithm Design and Analysis Process. Con't

3. Algorithm Design and Data Structure

- ▶ Programs = Algorithms + Data Structures
- ▶ Data structures: List, Array, Dictionary, Set, Queue, Stack, etc

4. Prove correctness of the algorithm

- ▶ Algorithm yields required **output** for every **legitimate input** in **finite** time
- ▶ Technique for proving correctness of;
 - ▶ **Exact algorithm** Mathematical induction can be used due to natural sequence of steps in an algorithm
 - ▶ **Approximate algorithm** show that the error produced by the algorithm does not exceed a predefined limit

Algorithm Design and Analysis Process. Con't

5. Analyzing an Algorithm

- ▶ **Time efficiency:** how fast it runs
- ▶ **Space efficiency:** How much extra memory it uses
- ▶ **Simplicity:** easier to understand, usually contains fewer bugs, sometimes simpler is more efficient, but not always!
- ▶ **Generality:** of the problem algorithm solves or the set of inputs it accepts

6. Coding Algorithm Write in a programming language for a real machine

- ▶ Standard tricks:
 - ▶ Compute loop invariant (which does not change value in the loop) outside loop
 - ▶ Replace expensive operation by cheap ones
 - ▶ E.g., check if n is even $n\%2 == 0$ is more expensive than $(n \& 1) == 0$
- ▶ Algorithm's **optimality**. *Actually, this question is not about the efficiency of an algorithm but about the complexity of the problem it solves: min effort exert to solve a problem.*

Important Problem Types

- ▶ Sorting
- ▶ Searching
- ▶ String Processing
- ▶ Graph Problem
- ▶ Combinatorial Problem
- ▶ Geometric Problem
- ▶ Numerical Problem

Sorting

- ▶ The *sorting problem* is to rearrange the items of a given list in nondecreasing order.
- ▶ Properties of sorting Algorithms
 - ▶ An algorithm is said to be **stable** if it preserves the **relative order** of any **two equal elements** in its input
 - ▶ An algorithm is said to be **in-place** if it does not require **extra memory**, except, for a few memory units
- ▶ Statement of problem: rearrange the number in a given order
 - ▶ Input: a sequence of N numbers $\langle a_1, a_2, \dots, a_N \rangle$
 - ▶ Output: a reordering of the input sequence $\langle a'_1, a'_2, \dots, a'_N \rangle$ so that $a'_i \leq a'_j$ whenever $i < j$
 - ▶ Instance: the sequence $\langle 5, 3, 2, 8, 3 \rangle$ becomes $\langle 2, 3, 3, 5, 8 \rangle$ after sorting
- ▶ Algorithm: Bubble sort, Selection sort, Insertion sort, Merge sort, Quick sort

Searching

- ▶ The *searching problem* deals with finding a given value, called a search key, in a given set.
- ▶ Statement of problem: search a key number in a given set
 - ▶ Input: a sequence of N numbers $\langle a_1, a_2, \dots, a_N \rangle$ search key $\langle k \rangle$
 - ▶ Output: the key $\langle k \rangle$ from the input sequence $\langle a'_1, a'_2, \dots, a'_N \rangle$
 - ▶ Instance: the sequence $\langle 5, 3, 2, 8, 3 \rangle$ search for 2 becomes return 2 or null if 2 doesn't exist
- ▶ Algorithm: sequential search, binary Search

String Processing

- ▶ A **string** is a sequence of characters from an alphabet.
- ▶ **String matching.** Is a particular problem of searching for a given word in a text.
- ▶ **Text strings:** letters, numbers, and special characters
- ▶ **Bit strings:** sequence of 0's and 1's
- ▶ **Gene sequence:** string of characters from four-character alphabet {A, C, G, T} for Adenine, Cytosine, Guanine, and Thymine, constituents of DNA
- ▶ **String matching:** search for a given word/pattern in a text or gene sequence

Graph Problem

- ▶ A graph can be thought of as a collection of points called **vertices**, some of which are connected by line segments called **edges**.
- ▶ The **traveling salesman problem (TSP)** is the problem of finding the shortest tour through n cities that visits every city exactly once. **Shortest-path**
 - ▶ Circuit board and VLSI chip fabrication, X-ray crystallography,
- ▶ The **graph-coloring problem** seeks to assign the smallest number of colors to the vertices of a graph so that no two adjacent vertices are the same color.
 - ▶ event scheduling
- ▶ **Graph traversal** (how to reach all points in a network ?)
- ▶ **Topological sorting** (is a set of courses with prerequisites consistent?)

Combinatorial Problem

- ▶ These are problems that ask, explicitly or implicitly, to find a combinatorial object such as
 - ▶ a permutation,
 - ▶ a combination,
 - ▶ or a subset—that satisfies certain constraints.
- ▶ Generally speaking, combinatorial problems are the most difficult problems in computing, from both a theoretical and practical standpoint.
 - ▶ the traveling salesman problem
 - ▶ graph-coloring problem

Geometric Problem

- ▶ **GP** Deal with geometric objects such as points, lines, and polygons.
- ▶ The ancient Greeks were very much interested in developing procedures for solving a variety of geometric problems,
 - ▶ Including problems of constructing simple geometric shapes triangles, circles, and so on with an unmarked ruler and a compass.
- ▶ The **closest-pair** problem is self-explanatory: given n points in the plane, find the closest pair among them.
- ▶ The **convex-hull problem** asks to find the smallest convex polygon that would include all the points of a given set.

Numerical Problem

- ▶ **NP** are problems that involve mathematical objects of continuous nature:
 - ▶ solving equations and systems of equations,
 - ▶ computing definite integrals,
 - ▶ evaluating functions, and so on.
- ▶ Principal difficulty stems from the fact that such problems typically **require manipulating real numbers**, which can be represented in a computer only **approximately**
- ▶ Since new applications require primarily algorithms for information storage, retrieval, transportation through networks, and presentation to users
 - ▶ **NP has lost its place in the computing industry**

Fundamental Data Structures

- ▶ Linear Data Structures
 - ▶ Array, Linked List, Stacks and Queues
- ▶ Graphs
- ▶ Trees

Linear Data Structures: Array

- **Array:** A sequence of n items of the **same data type**, stored **contiguously in memory** and made accessible through **array indices**.

indices →	0	1	2	3	4	5	6	7
A:	20	2	11	23	84	30	6	17

e.g.,

$A[4] = 84$,

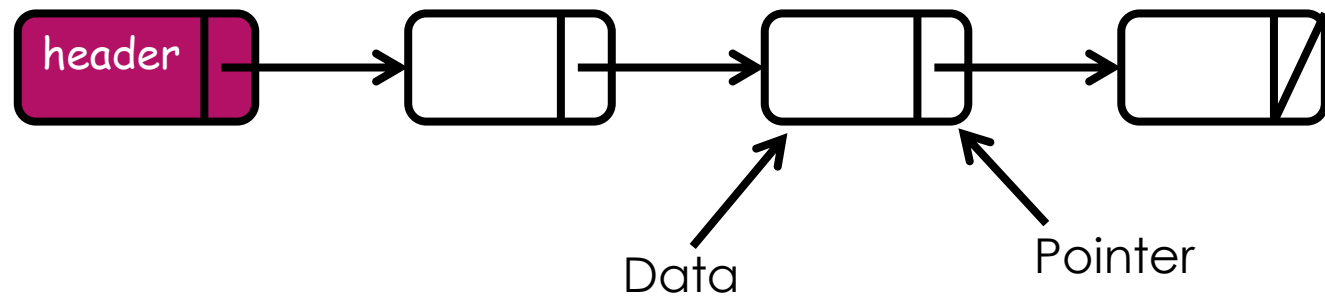
$A[2] = 1$,

$A[7] = 17$

Linear Data Structures: Linked List

► Linked List

- A sequence of nodes with links (called pointers) to neighbors
- Singly or Doubly linked lists



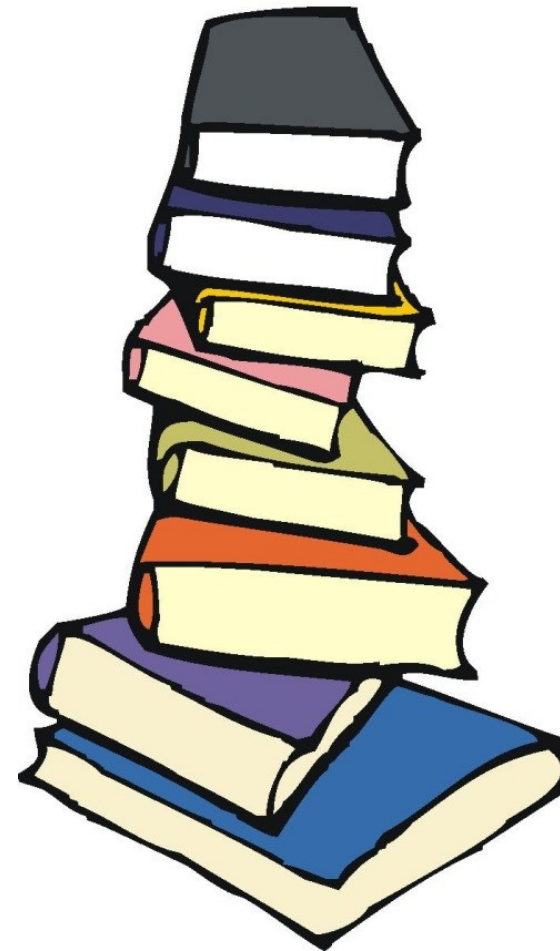
Linear Data Structures: Array vs Linked List

Differences between Array and Array List

Features	Array	Array List
Access time	Fixed	Variable
Update Cost	Higher	Lower
Storage Anticipation	Needed	Not Needed

Linear Data Structures: Stack

- ▶ Stack
 - ▶ Insertion/deletion only at the top
 - ▶ Last In First Out (LIFO)
 - ▶ Two operations: push and pop
 - ▶ E.g., recursion needs stack
 - ▶ `java.util.Stack`



Linear Data Structures: Queue

- ▶ Queue
 - ▶ Insertion from rear, deletion from front
 - ▶ First In First Out (FIFO)
 - ▶ Two operations: enqueue, dequeue

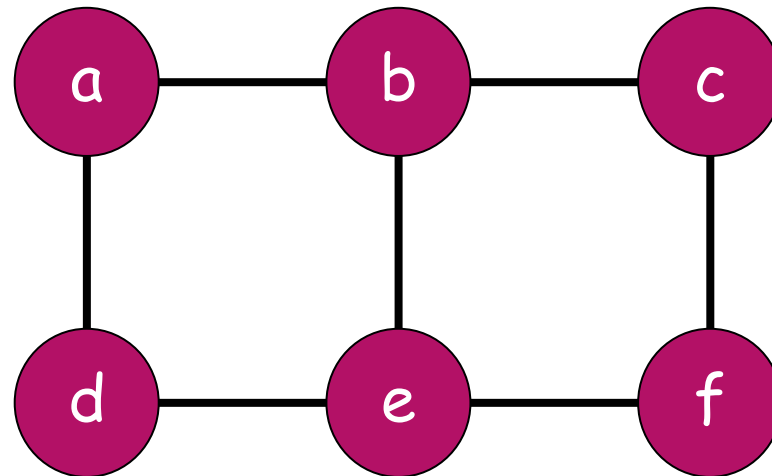


Linear Data Structures: Queue...

- ▶ Priority queues
 - ▶ Select item of highest priority among dynamically changing candidates. E.g., schedule job in a shared computer
 - ▶ Principal operations:
 - ▶ Find largest, delete largest, add new element
 - ▶ Add/delete must result in priority queue
 - ▶ Array or Sorted array based implementation is inefficient
 - ▶ “**Heap**” is better (We shall discuss in Transform and Conquer approach)

Graph

- ▶ A graph, $G = \langle V, E \rangle$ is defined by a pair of sets:
 - ▶ V is called vertices and E is called edges

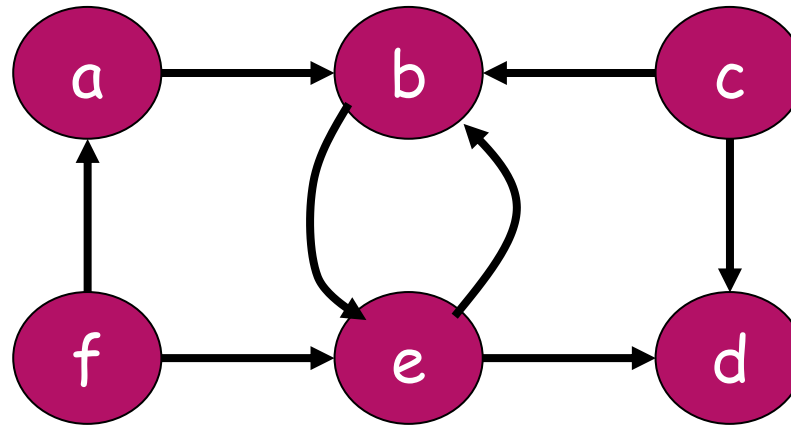


$$V = \{a, b, c, d, e, f\}$$

$$E = \{ (a, b), (b, c), (c, f), (f, c), (e, d), (d, a), (b, e) \}$$

Graph

- ▶ Undirected graphs and Directed Graphs (Digraphs)
- ▶ We usually will not allow graphs to have loops
- ▶ What is the **maximum number of edges** in an undirected graph with $|V|$ vertices?



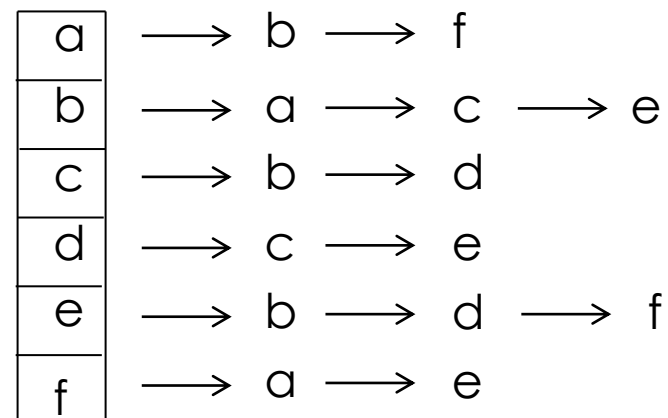
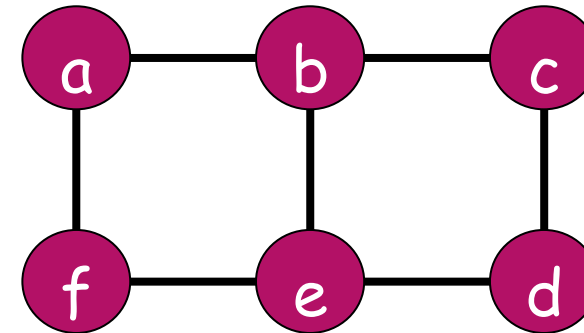
Directed graph

Representing Graphs

- ▶ Adjacency matrix
 - ▶ $N \times N$ boolean matrix
 - ▶ $A[i, j] = 1$ if there is an **edge** from i-th vertex to j-th vertex
 - ▶ Adjacency matrix of an undirected graph is symmetric.
- ▶ Adjacency list
 - ▶ A collection of linked lists, one for each vertex, that contain all neighbors of a vertex

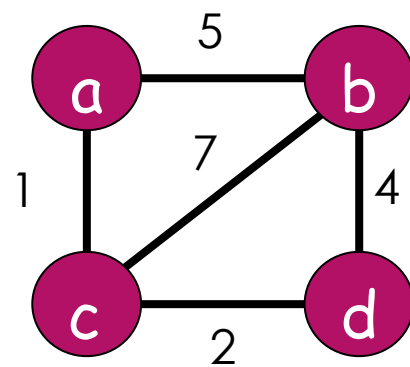
Representing Graphs

	a	b	c	d	e	f
a	0	1	0	0	0	1
b	1	0	1	0	1	0
c	0	1	0	1	0	0
d	0	0	1	0	1	0
e	0	1	0	1	0	1
f	1	0	0	0	1	0



Weighted Graphs

- Edges have numbers associated with them, e.g., distance between two cities



	a	b	c	d
a	∞	5	1	∞
b	5	∞	7	4
c	1	7	∞	2
d	∞	4	2	∞

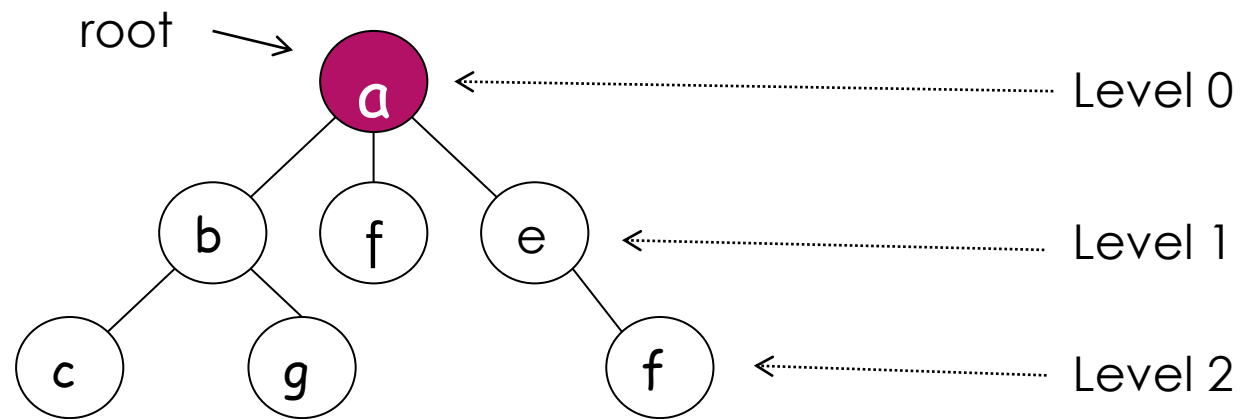
a	→ b, 5 → c, 1
b	→ a, 5 → c, 7 → d, 4
c	→ a, 1 → b, 7 → d, 2
d	→ b, 4 → c, 2

Tree

- ▶ A tree is a connected acyclic graph. $|E| = |V| - 1$
- ▶ Forest: has no cycle but may be disconnected
- ▶ Between every two vertices there is exactly one simple path.
- ▶ Any vertex can be regarded as the root,
- ▶ and then we have a rooted tree.

Tree...

- Ancestors: For vertex v , all the vertices on the simple path from root to v are called v 's ancestors.

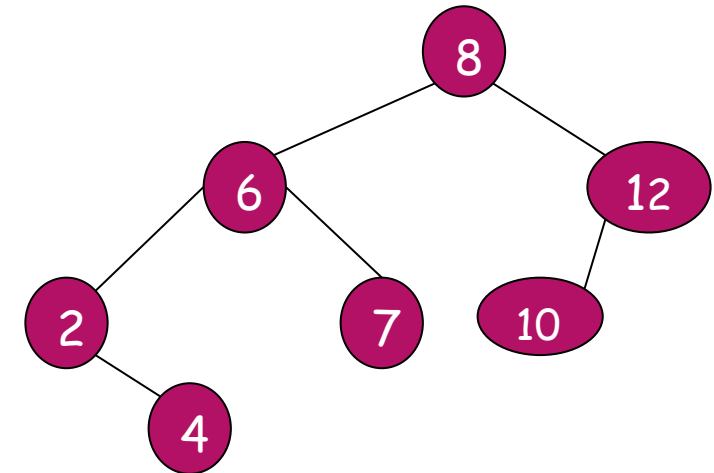


Tree...

- ▶ **Descendants:** All vertices for which v is an ancestor are descendants of v .
- ▶ **Parent, child, sibling:**
 - ▶ If (u, v) is the last edge of the simple path from root to v , then u is *parent* of v and v is a *child* of u .
 - ▶ Vertices having same parents are siblings.
- ▶ **Leaves:** A vertex without children is a leaf.
- ▶ **Subtree:** v with all its descendants is the subtree rooted at v .
- ▶ **Depth of vertex:** Length of the simple path from root to v is v 's depth.
- ▶ **Height of tree:** The length of the longest simple path from root to leaf.

Ordered Tree

- ▶ An ordered tree is a rooted tree in which all children of each vertex are ordered.
- ▶ **Binary tree:**
 - ▶ An ordered tree, each vertex has at most two children and each children is designated as either left child or right child.
 - ▶ $\lfloor \log_2 n \rfloor \leq h \leq n - 1$, here n is the number of nodes in the tree and h is tree's height
- ▶ **Binary Search Tree:**
 - ▶ Each vertex is assigned a number
 - ▶ A number assigned to a parental vertex is larger than all numbers in the left subtree and smaller than all numbers in the right subtree.



What we discussed about

- ▶ What is an Algorithm?
 - ▶ Properties of Algorithms
- ▶ Algorithm design and analysis process.
 - ▶ How to Understand program development problem
 - ▶ Algorithm Design Techniques
 - ▶ ...
- ▶ Important Problem Types
- ▶ Fundamental Data Structure

Thank You