THE UNIVERSITY OF UTAH
ASIA CAMPUS
SOUTH KOREA

Department of Electrical and Computer Engineering

Spring 2022

Introduction to Algorithms and Data Structure (CS 2420)

1. Introduction

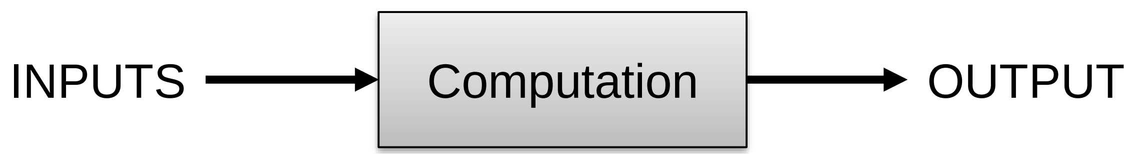
Habtamu Minassie
Habtamu.aycheh@utah.edu

Contents

- Basic Concepts of algorithms
- Basic concepts of data structure
- Abstract Data Types (ADT)
- Why Algorithms and Data Structure?

What is an algorithm?

- An algorithm is a finite set of instructions that are carried in a specific order to perform a specific task.
  - A step –by-step procedure for solving problems
- A computational problem specifies an input-output relationship
  - What does the input look like?
  - What should the output be for each input?



  - Example 1
    - Input: an integer number N
    - Output: is the number prime? Yes or No
  - Example 2
    - Input: a list of names of people
    - Output: List of names sorted alphabetically

Requirements of an algorithm

- Input (≥ 0) : Zero or more inputs
- Output (>0) : At least one output
- Clear and unambiguous: specify every step completely, so a computer can implement it without any further "understanding"
- Correct: for each input, produce an appropriate output
- Efficient: run as quickly as possible, and use as little memory as possible
- Finite : It should terminate after a finite number of steps.

Some Algorithm Design Techniques

- Depending on the strategy for solving a problem , algorithms are classified as follows:
  - Divide –and –Conquer Algorithms
    - A given problem is fragmented into subproblems which are solved partially
      - Frequently used in searching and sorting algorithms
  - Greedy algorithms
    - An immediately available best solution at each step is chosen
      - Useful in graph theory
  - Back–tracking algorithms
    - All possible solutions are explored , until the end is reached and the steps are traced back.
  - Dynamic Programming
    - An organized way to find an optimal solution by systematically exploring all possibilities without unnecessary repetition
      - we need to make the optimal (lowest cost, highest value, shortest distance, and so on) choice among a large number of alternative solutions

What is Data Structure?

- Data structure is a way to store, organize and manage data in computer system so that it can be used efficiently.
- Data structure can be viewed as :
  - **Mathematical/logical model**
    - Abstract Data Type (ADT)
  - **Implementation**

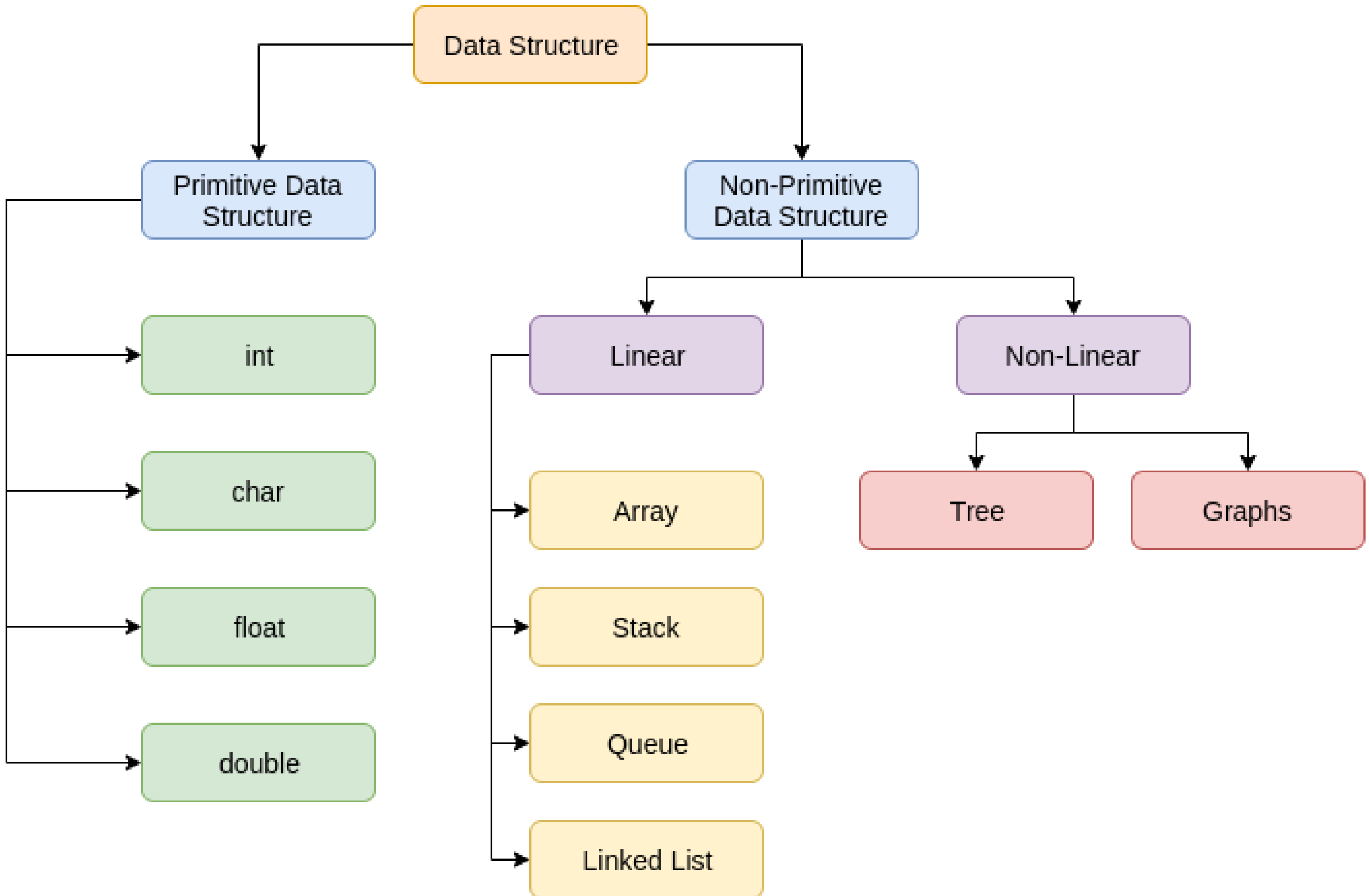
Data Abstraction

- What is a data type?
  - It is a collection objects and a set of operations that act on those objects
  - Example: int data type:
    - Values: {INT_MIN, …,-2, -1, 0, 1, 2,…, INT_MAX}
    - Operations on integers: {+, -, %, *, /, …}
  - Primitive data type
    - Built into the language
      - Example: int, char, short, long, float, double,…

Abstract Data Type (ADT)

- An ADT is a mathematical model(logical view) of a data structure that specifies
  - The type of data stored
  - The operations supported on them
- Examples of ADTs include: List, Stack, Queue, Tree, Graph, etc.
- An ADT specifies what each operation does, but not how it does
- ADT can be implemented using one of many different data structures
  - Example: **Stack** can be implemented using Array or **Linked list**.
- **Data Structures = ADT + Implementation**

Classification of Data Structures



Data Structure and Memory Allocation

- Memory allocation can be classified as:
  - Contiguous memory allocation
    - Arrays
  - Non-Contiguous memory allocation
    - Linked List

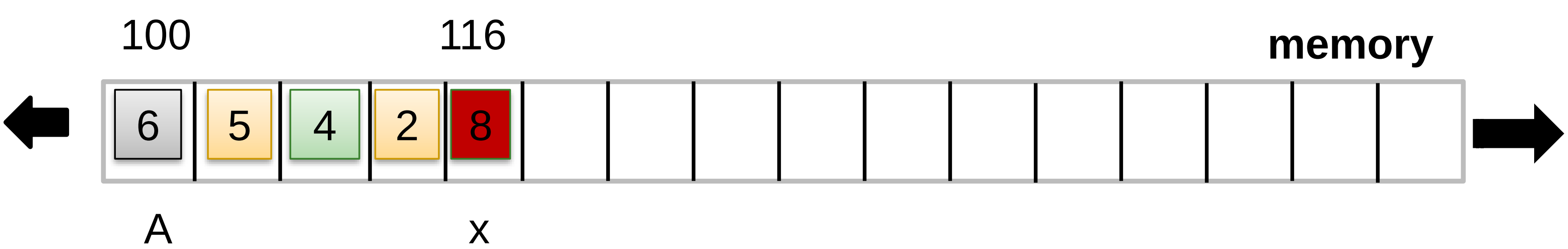
Contiguous Memory Allocation

- An array stores n objects in a single contiguous space of memory
  - **Static Array** – the size is fixed
  - **Dynamic Array** – the size is variable
- **Static Array**
  - Impossible to reallocate new memory location than specified

```
//Example
int x =8;
int A[4] = {6,5,4,2};
```

int A[max]

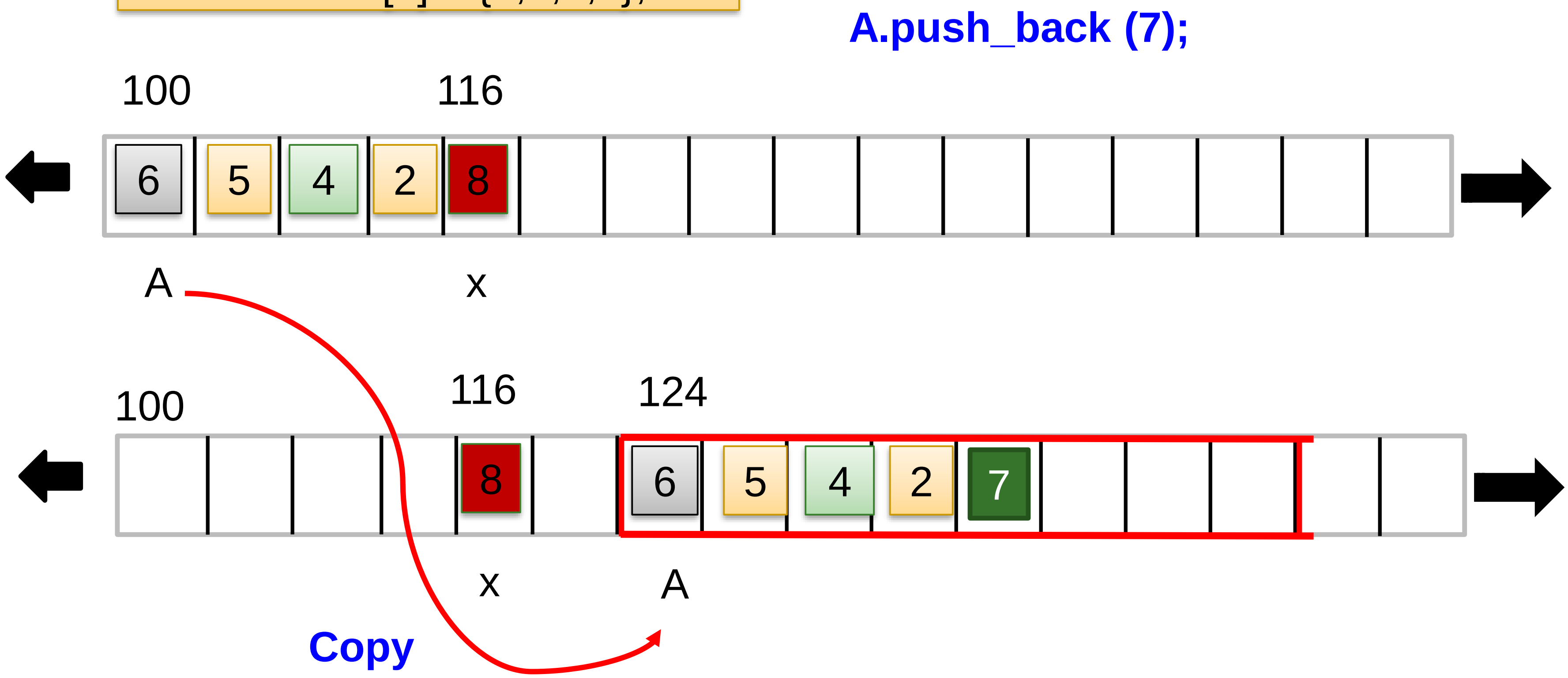
Max=?



Dynamic Array

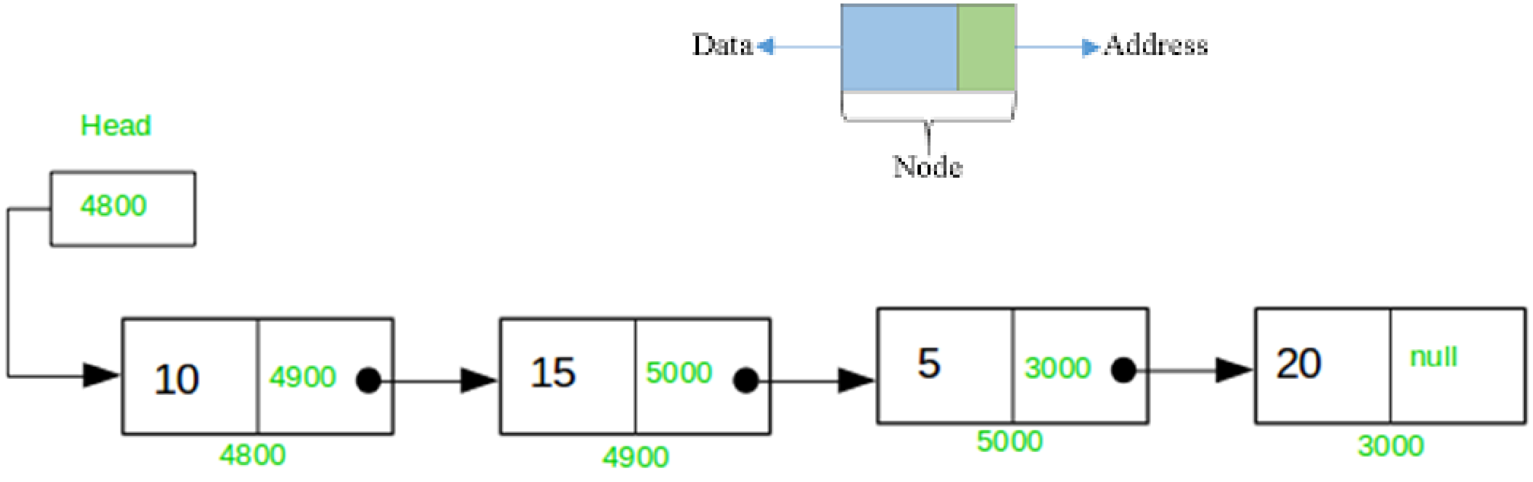
- If more memory is required, a request for new memory usually requires copying all information into a new memory location

```
//Example
int x {8};
vector<int> A[4] = {6,5,4,2};
```



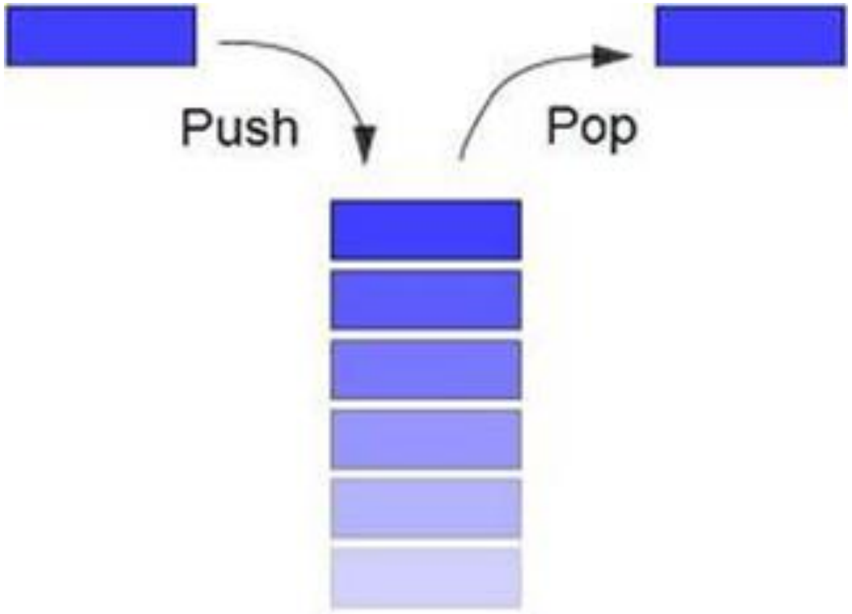
Non-contiguous allocation

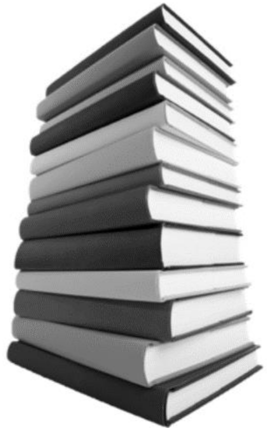
- Memory is not necessary to be contiguous
- Linked list associates two pieces of data with each item being stored:
  - Data object
  - A reference pointing to the next node

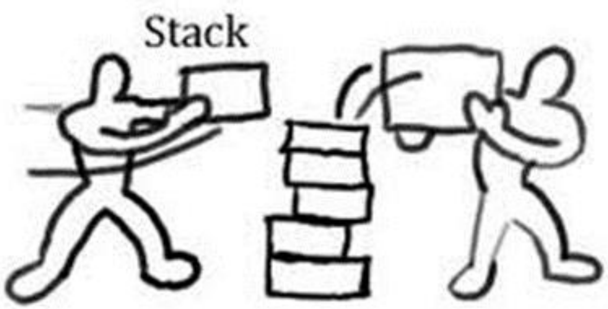


Stack

- LIFO linear data structure

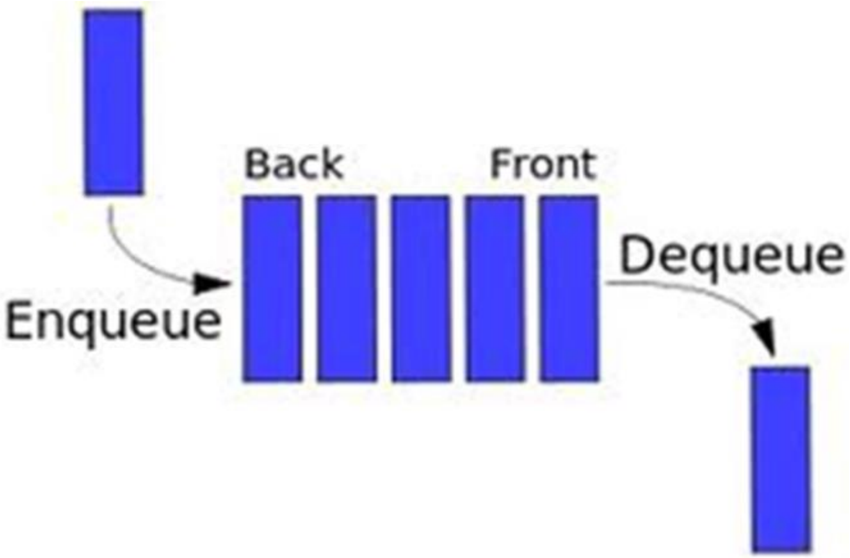






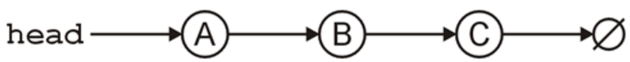
Queue

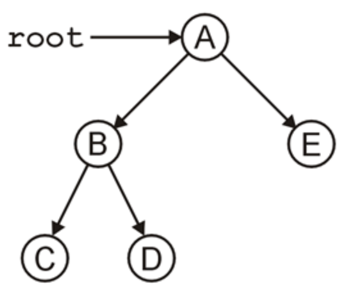
- FIFO linear data structure



Tree

- Non –linear data structure
- A tree is a variation of a linked list
  - Each node points to an arbitrary number of subsequent nodes
  - Useful for storing hierarchical data
  - We will see that it is also useful for storing sorted data
  - Binary Tree
    - Trees where each node points to at most two other nodes

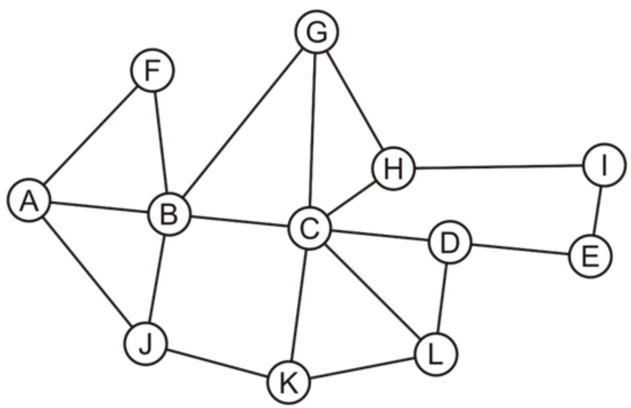


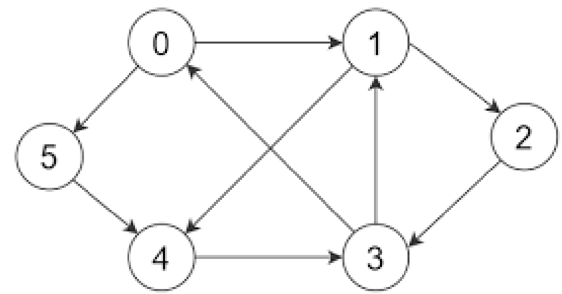


Graph

- Graph is a non-linear data structure that allows arbitrary relations between any two objects in a container
- A graph G is an **ordered /unordered pair** of a set V of vertices and a set E of edges

Graph G = <V,E>



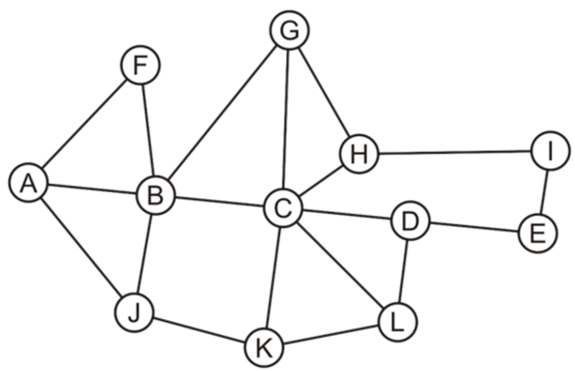


Ordered pair : <A,B>≠ <B,A> if A ≠B – Directed Graph

Unordered pair : {A,B} ={B,A} – undirected graph

Graph Representation

- Adjacency Matrix
  - Represented using a two-dimensional array
  - For example, consider the network
    - 12 vertices (nodes)
    - 19 edges (node-to-node connections)

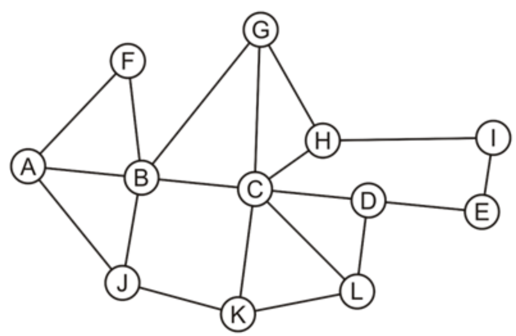


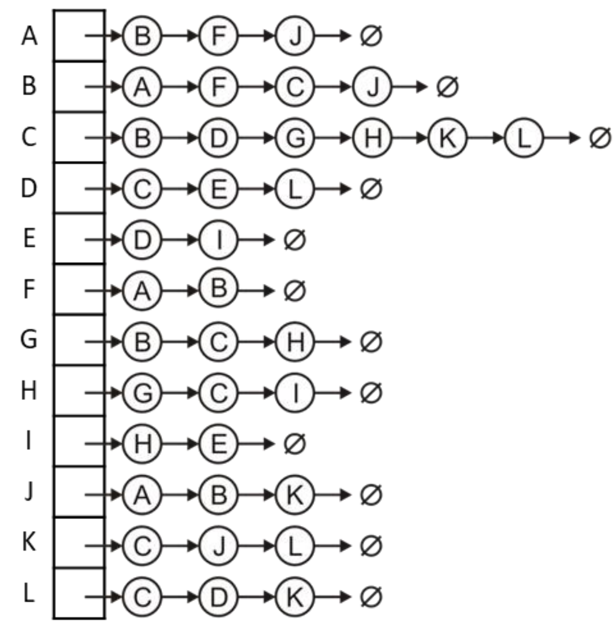
$$A_{ij} = \begin{cases} 1, & \text{if } \exists \text{ edge from } i \text{ to } j \\ 0, & \text{otherwise} \end{cases}$$

	A	B	C	D	E	F	G	H	I	J	K	L
A		1				1				1		
B	1		1			1	1			1		
C		1		1			1	1			1	1
D			1		1							1
E				1					1			
F	1	1										
G		1	1					1				
H			1				1		1			
I					1			1				
J	1	1									1	
K			1							1		1
L			1	1							1	

Graph Representation

- **Adjacency List**
  - Alternatively, uses a hybrid method
  - An array of linked lists

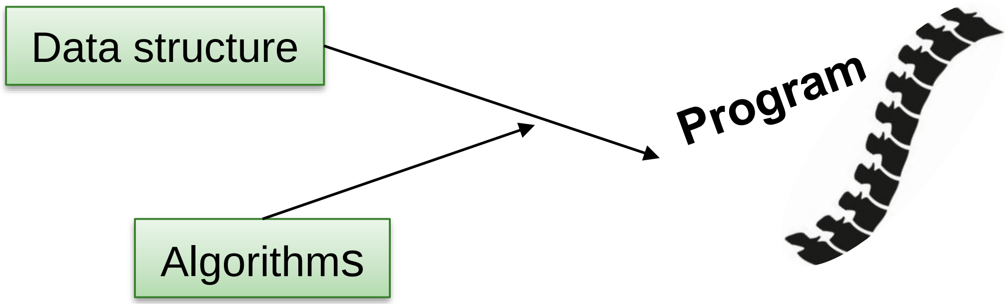




Summary

- Why Algorithms and Data Structure?

  - Computer programming is all about problem solving



  - Algorithms make use of data structures and data structures need algorithms to function
  - Data structures are essential ingredients in creating fast and powerful algorithms.
  - The programmer must be able to write the programs in such a way that the program should take optimum memory space and increase its execution speed
  - **Data Structure + Algorithms = Program**

  - There is no ultimate data structure
  - The choice depends on our requirements


Use the right tool for the right job
thank you