华南理工大学

South China University of Technology

Intro to Computer Science and Software Engineering

Data Structures and Abstract Data Type

Dr Yubei Lin
yupilin@scut.edu.cn
School of Software Engineering

Why Data Structures

- In solving complex problem, instead of considering a single entity individually, we often deal with a set of related items.
- A data structure uses a collection of related variables that can be accessed individually or as a whole.
  - i.e. dealing with a set of data items with a specific relationship among them
- The logical data organization vs the memory layout

Three basic data structures

- We look at three basic data structures:
  - arrays
  - records
  - linked lists.

- Most programming languages have an implicit implementation of arrays and records, and use pointers and records to simulate linked lists.

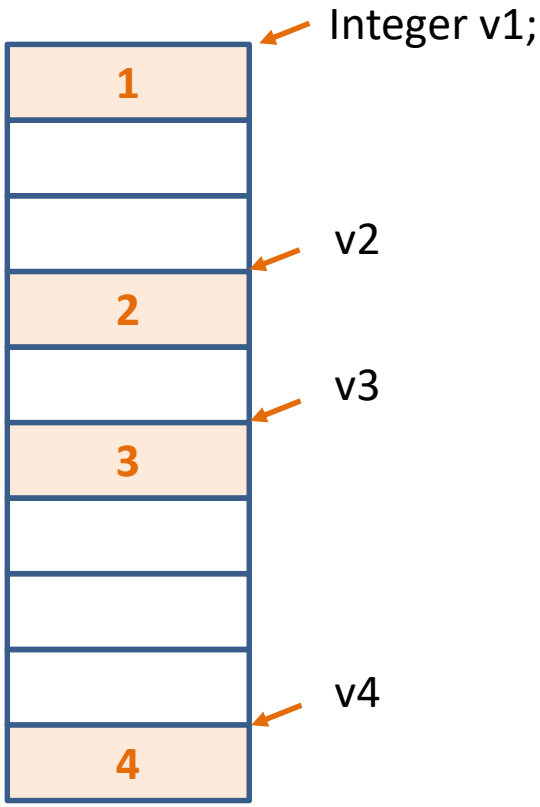
Arrays

- Considering dealing with 20 numbers: read, process and print them
  - Define 20 variables, and treat each number individually, which might be acceptable
  - But, what about 3000 numbers
- An array is a fixed-size, sequenced collection of elements of the data type.
- With array, you can use subscripts and loops to process a large amount of data items(e.g. numbers) very easy.

Memory layout of an array

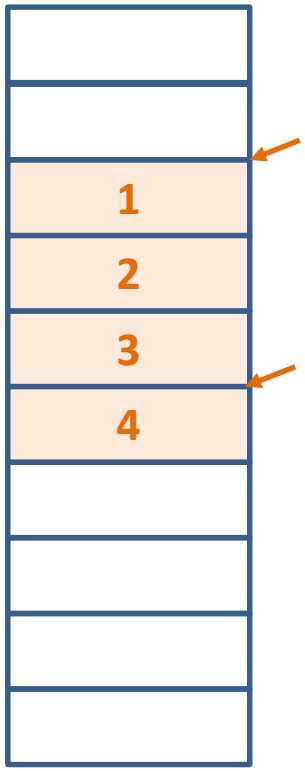
Single variables scatters in memory

- Individual name ~ address/pointer



Array: continuous memory allocation

- Single name ~ addresses



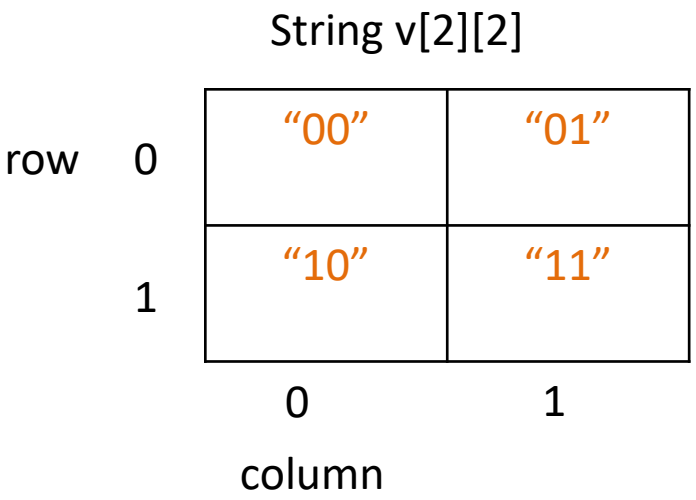
Accessing elements

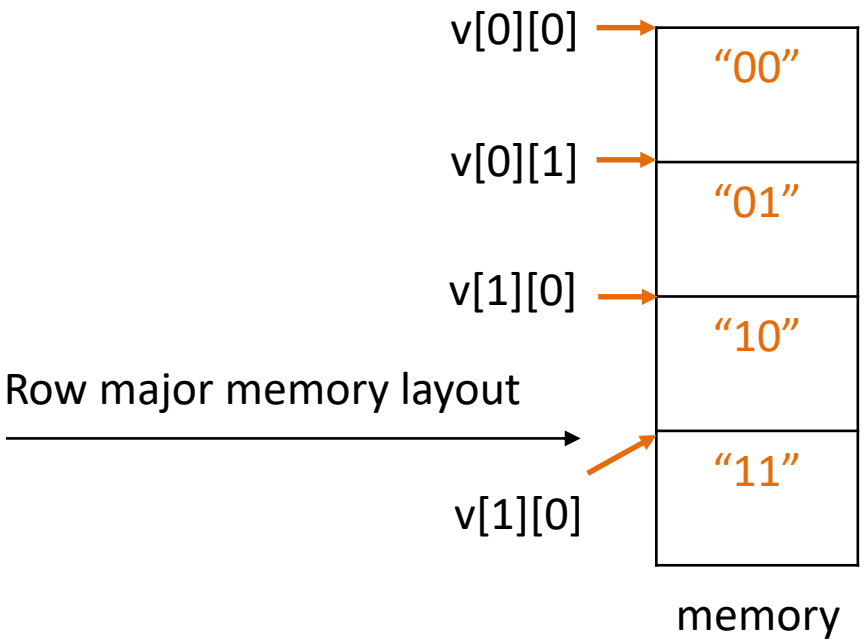
- subscript/indexing
- loops

One-dimensional array

Multidimensional arrays

- One-dimensional
  - Data are organized linearly in only one direction
- Two-dimensional
  - Data are organized in a table that consists of rows and columns.





Application of arrays

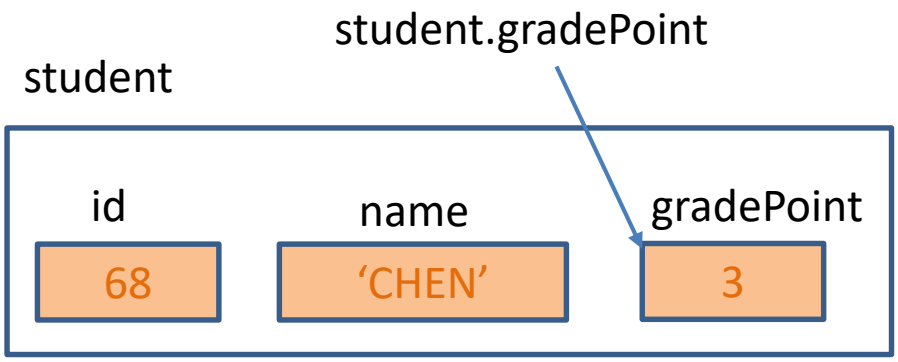
- The frequency array and its graphical representation.

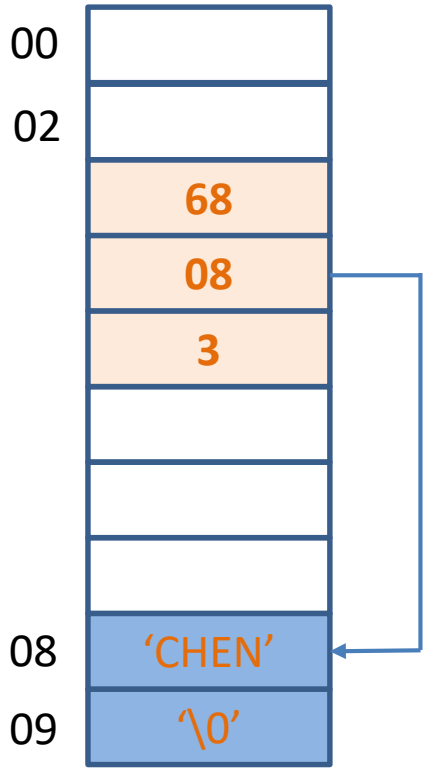
Records

- A record is a collection of related elements, possibly of different types, having a single name.
- Each element in a record is called a field!
- The difference~
  - Array: elements of the same type
  - Record: element of the same or different types
- Accessing individually fields
  - Using “record-name.field-name”

Memory layout of a record

```
Student{
    Integer id;
    String name; // pointer ~ address
    Integer gradePoint
};
Student student[3];
```





Linked lists

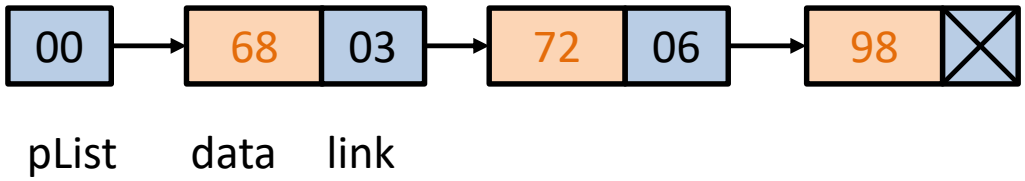
- A linked list is an **ordered** collection of data in which each element contains the location of the next element; that is, each element contain two parts: data and link.
- The elements in a linked list are called nodes
  - data part: the useful information/data to be processed
  - link part: the pointer (an address) that is used to chain the data together.
- Singly or doubly linked list

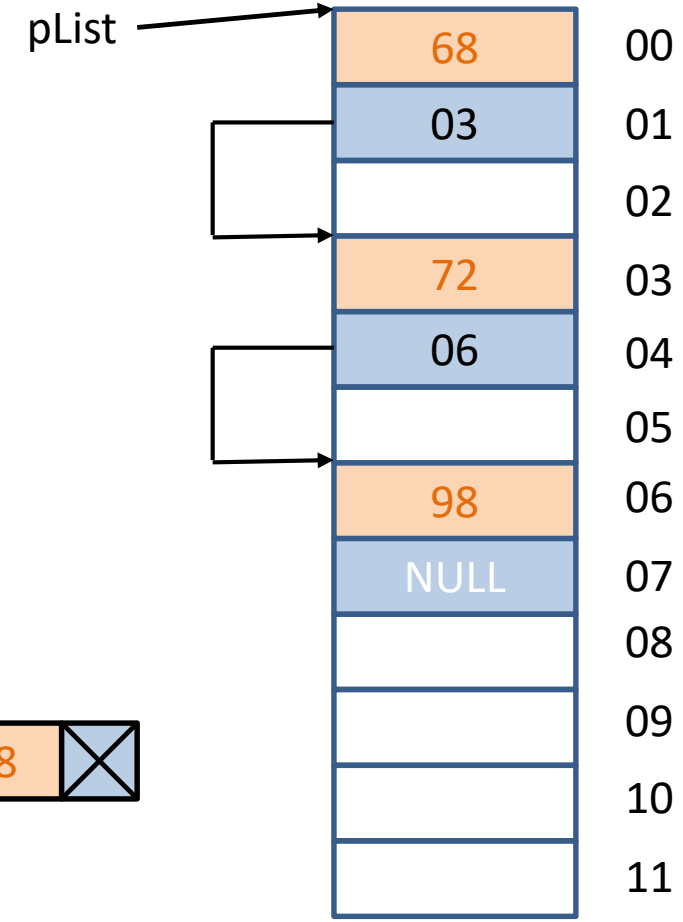
Pointers of linked list

- A linked list must always have a head pointer!
- More pointers needs for operations
  - pLoc: pointer to the node currently being located
  - pLast: pointer to the last

Memory layout of a linked list

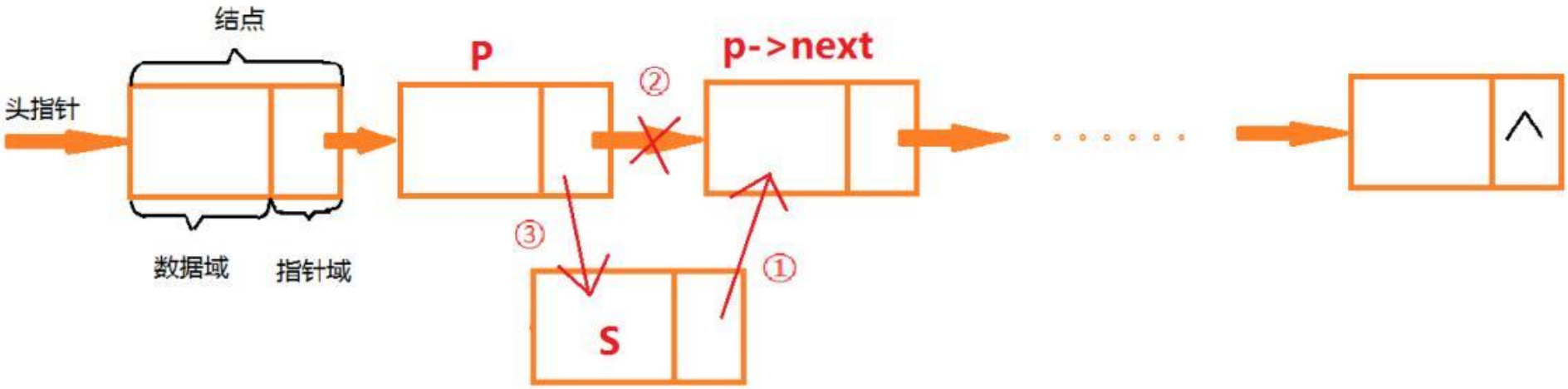
```
Record{
   integer data;
   Record * link;
}

Record * pList;
```





Operations on linked list

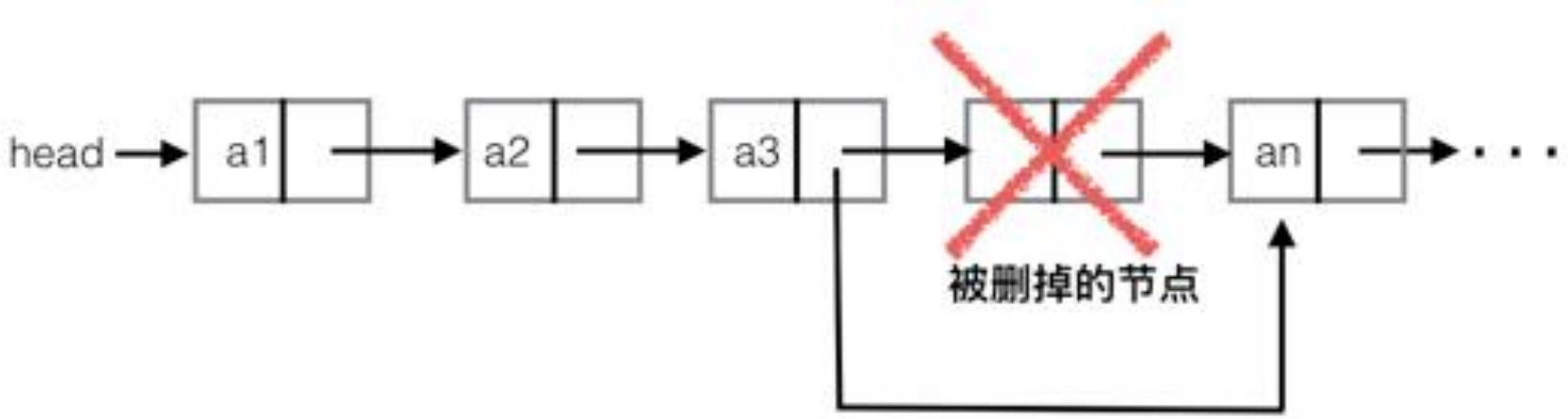
- Inserting



所有图片均来自网络

Operations on linked list

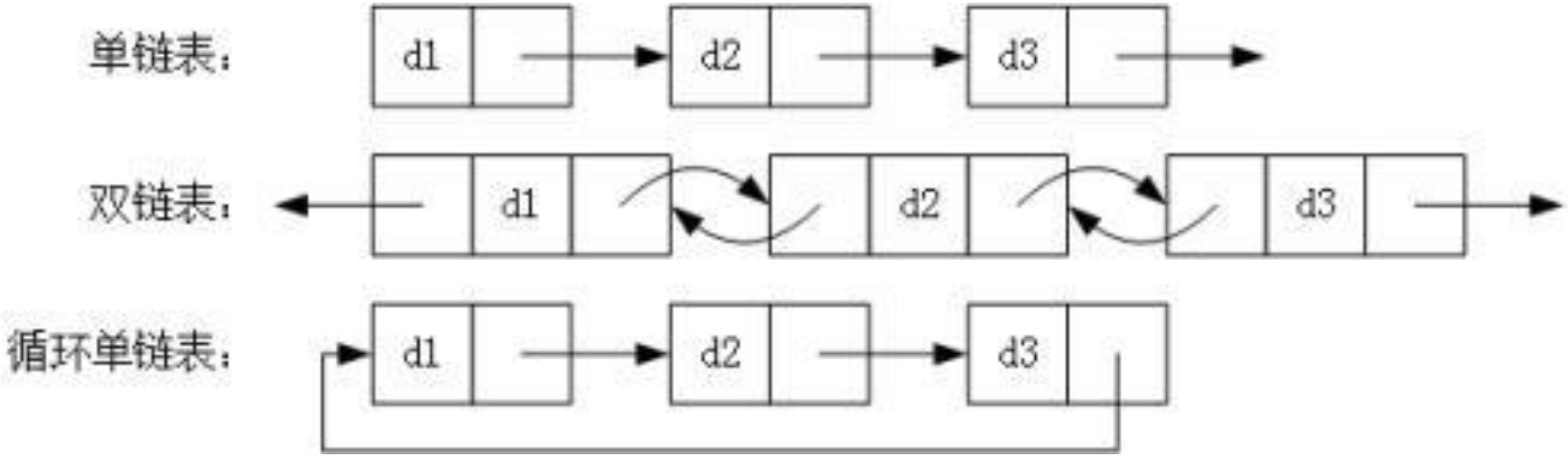
- Deleting



所有图片均来自网络

Operations on linked list

- Searching, retrieving and traversing a list



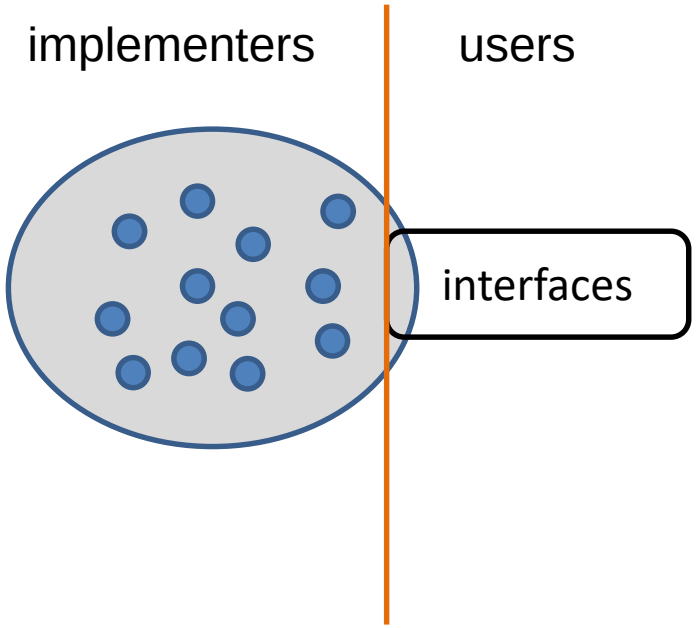
所有图片均来自网络

Question

- What about inserting or deleting an element of an array?
  - Easy or not?

Abstract Data Type

- The concept of abstraction means
  - You know what a data type can do
  - How it is done is hidden



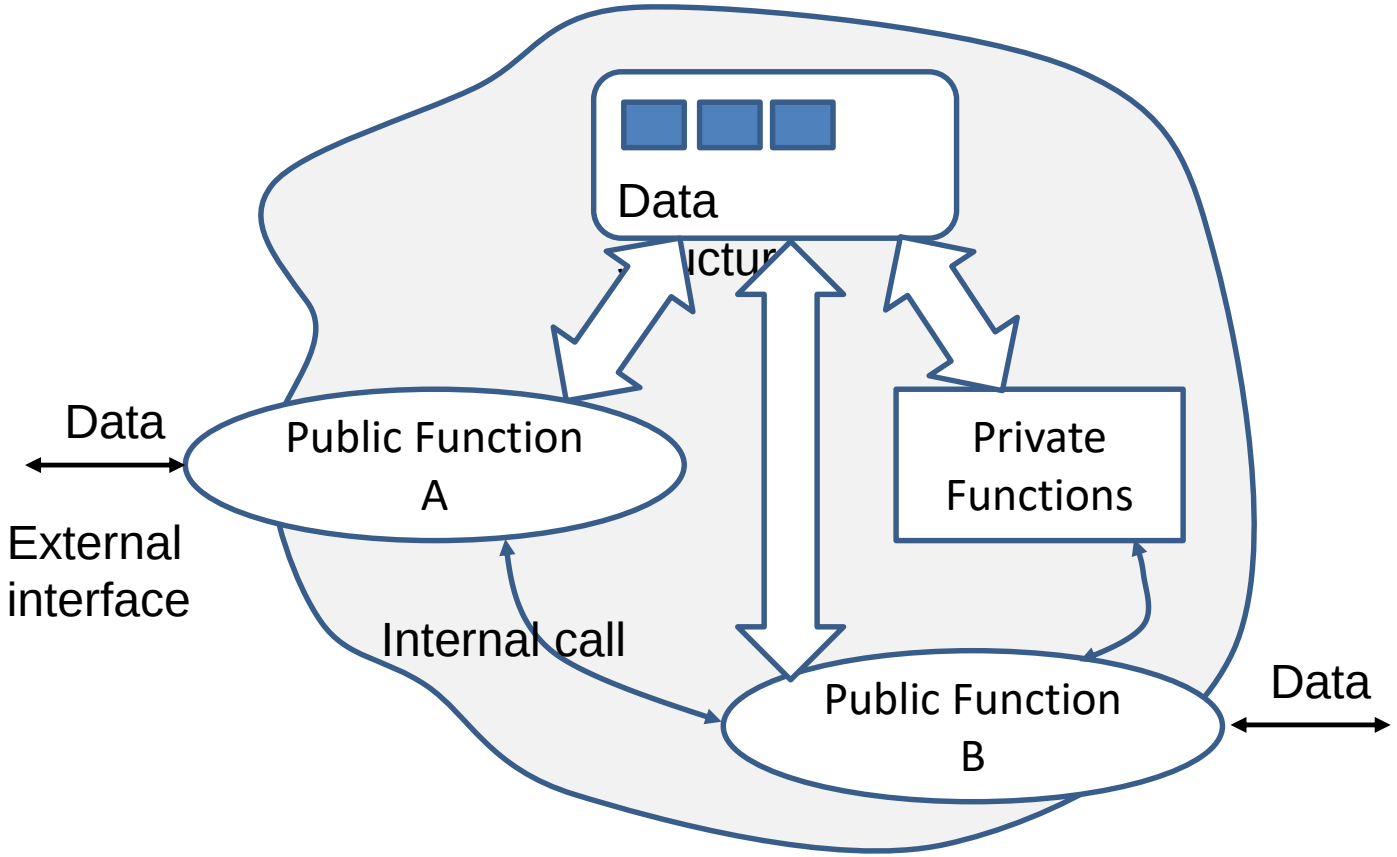
Abstract Data Type

- Abstract data type (ADT)
  - Declaration of data (Data structure)
  - Declaration of operations (Interfaces)
  - Encapsulation of data and operations (implementation is hidden)

ADT vs Data Structure

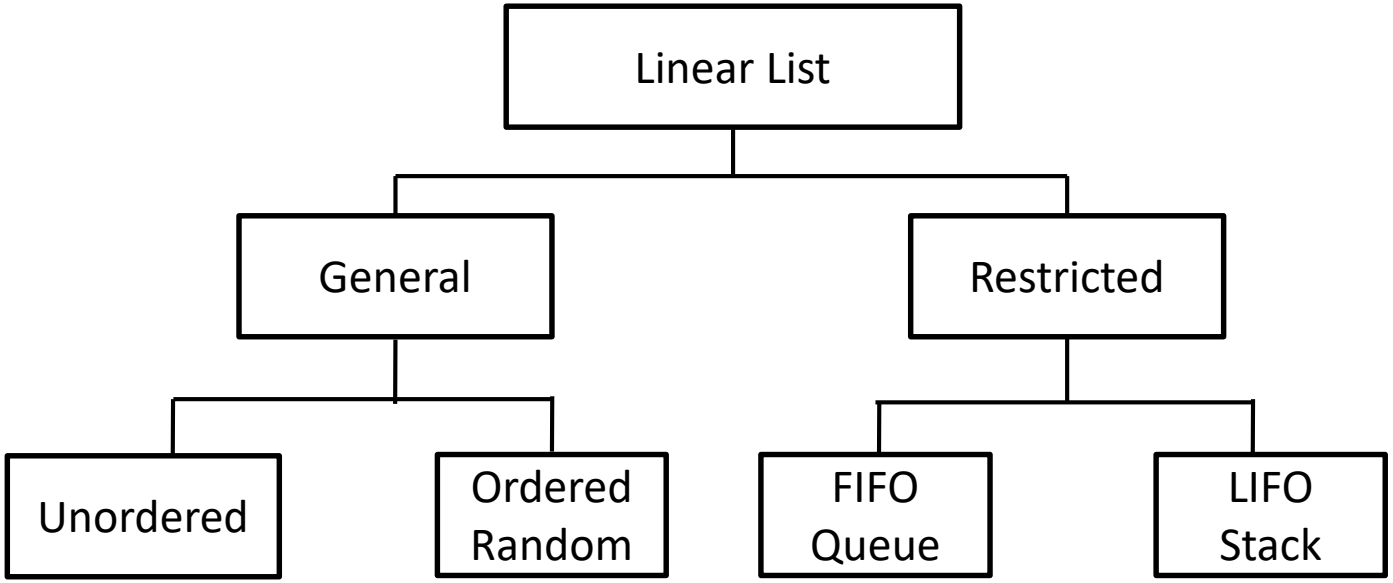
- **ADT** is a mathematical model for data types where a data type is defined by its behavior (semantics) from the point of view of a *user* of the data, specifically in terms of possible values, possible operations on data of this type, and the behavior of these operations.
- This contrasts with data structures, which are concrete representations of data, and are the point of view of an implementer, not a user.

Model of ADT



Linear lists

- A linear list is a list which each element has a unique successor.
  - a sequential structure



Linear lists

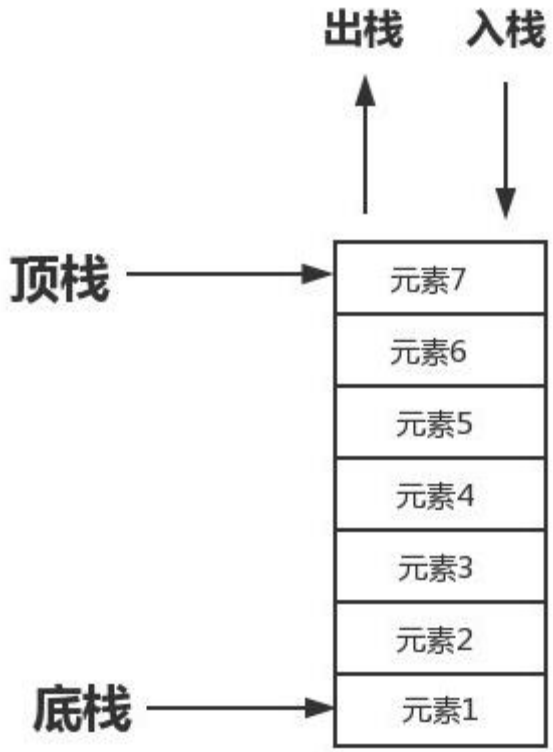
- General: data can be inserted or deleted anywhere, and there are no restriction on the operations that can be used to process the list.
  - Unordered and ordered
- Restricted: data can only be added or deleted at the ends of the list, and processing is restricted.
  - FIFO (queue) and LIFO (stack)

Operations on ordered linear list

- Insertion
  - When inserting, the ordering of the list must be maintained.
  - **Overflow**: no space for new element
- Deletion
  - **Underflow**: the list is empty
- Retrieval and traversal
- Implementation: by arrays and linked lists
  - For elements being accessed randomly (??)

Stacks

- A stack is a restricted linear list in which all addition and deletion are made at **the top**!
  - That is, Last in, First out (LIFO).
- Operations:
  - Push (insertion): add an element at the top of the stack;
  - Pop (deletion and retrieval): remove an element at the top of the stack, and return it to the user
- Implementation: commonly by linked lists



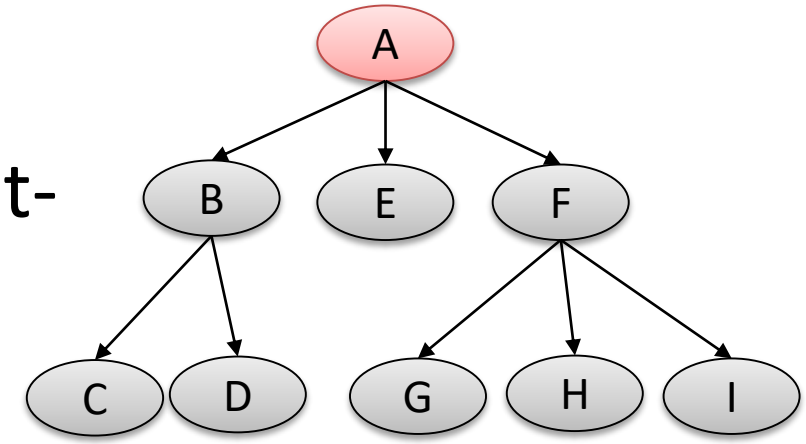
Queues

- A queue is a linear list in which data can be inserted at one end, called **the rear**, and deleted from the other end, called **the front**.
  - That is, First in, First Out (FIFO).
- Operations
  - Enqueue (insertion): at the rear
  - Dequeue (deletion): at the front
- Implementation: commonly linked lists.

队列的结构图

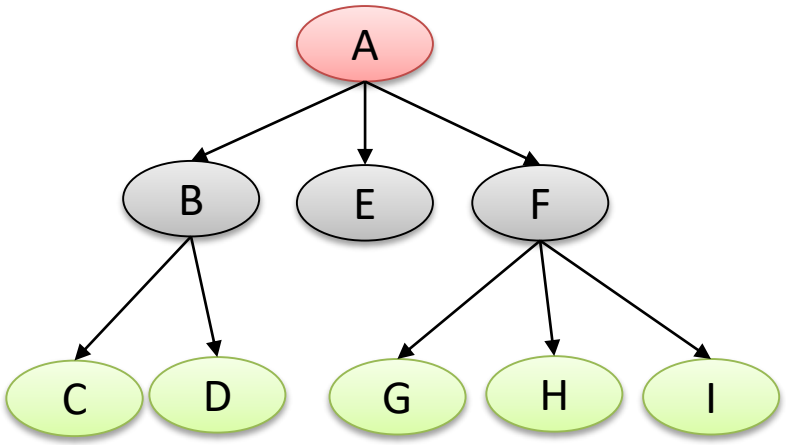
Trees: basic concepts

- Nodes and Braches
- Degree: in-degree and out-degree
- **Root** node:
  - The only node has an in-degree of zero!
- Others nodes must has an in-degree of exactly one!



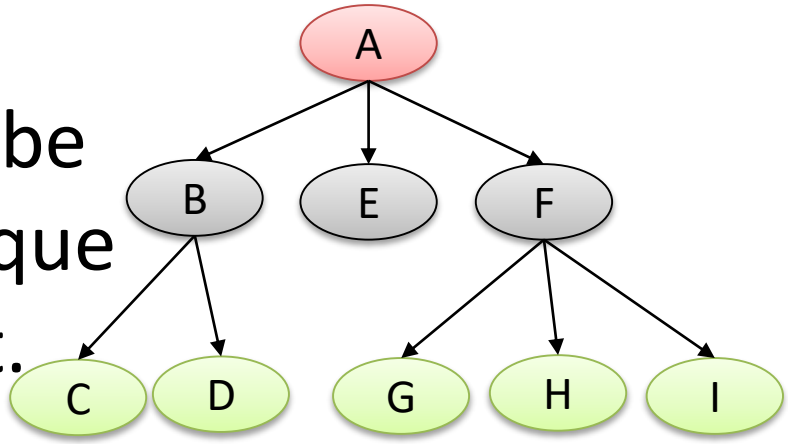
Trees: terminology

- Leaf: any nodes has an out-degree of zero
  - C, D, G, H and I
- Internal nodes:
  - B, F
- Parents
  - A, B and F
- Children
  - B, E, F, C, D, G, H and I
- Siblings: {B, E, F}, {C, D}, {G, H, I}
- Leaves: {C, D, E, G, H, I}
- Ancestors and Descendants



Trees: terminology

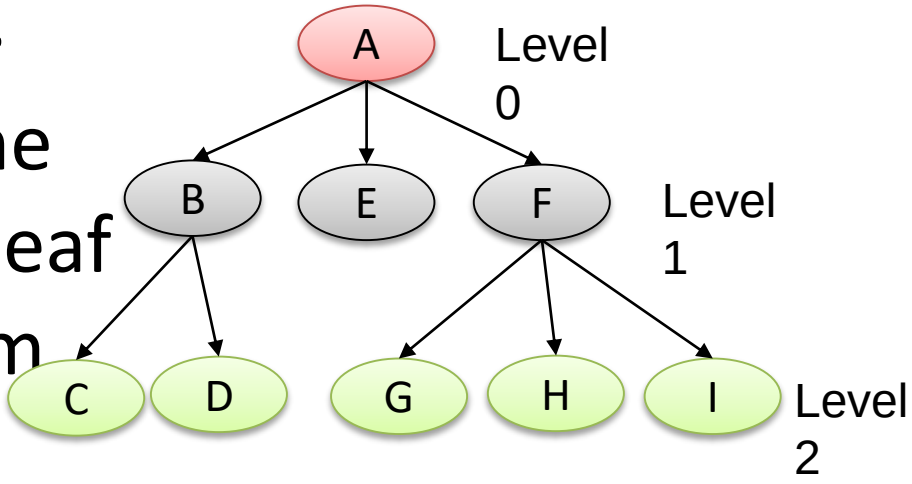
- Path: a sequence of nodes in which each node is adjacent to the next one.
  - e.g {A, B, C}, {A}, {A, B}
- Every node in the tree can be reached by following a unique path starting from the root.



所有图片均来自网络

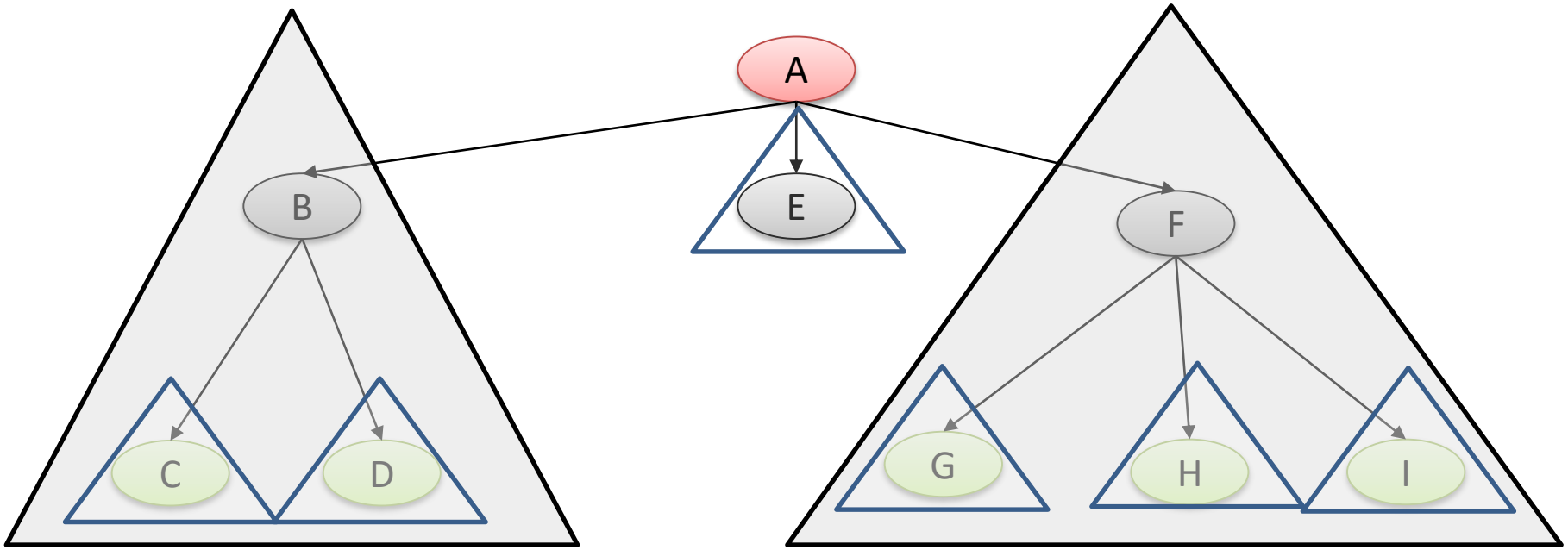
Trees: terminology

- The **level** of a node is its distance from the root.
- The **height/depth** of the tree is the level of the leaf in the longest path from the root plus 1.



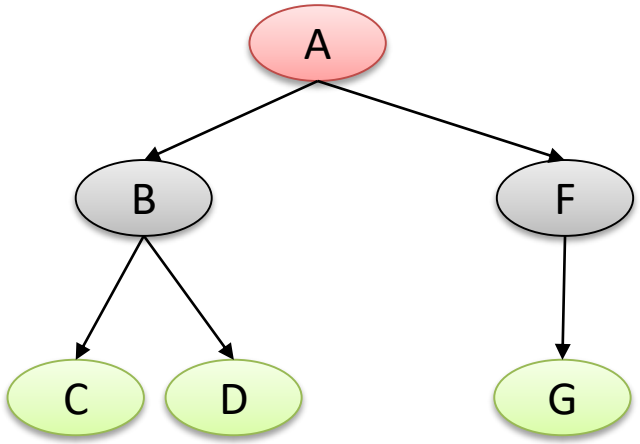
Trees: terminology

- Subtree: is any connected structure below the root!



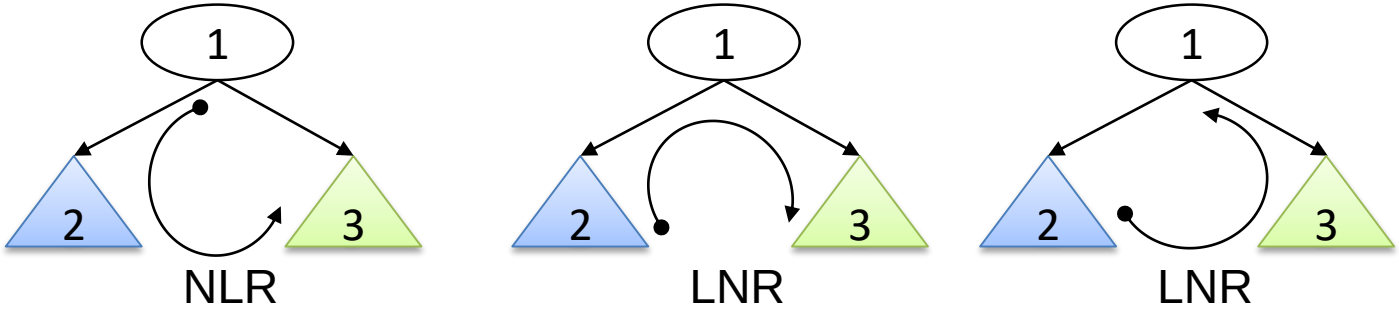
Binary trees

- A binary tree is a tree in which no node can have more than two subtree.
  - i.e, can have zero, one, or two~
- Properties
  - Height H, given N nodes
    - $H_{max} = N$; $H_{min} = [\log_2 N]+1$
  - Number of nodes N, given height H
    - $N_{min} = H$; $N_{max} = 2^H-1$
  - Balance B: $B = H_L - H_R$
    - A binary is balanced if the height of its subtrees differs by no more that 1 (i.e. B is -1, 0, or 1) and its subtrees are also balanced.



Binary trees: operations

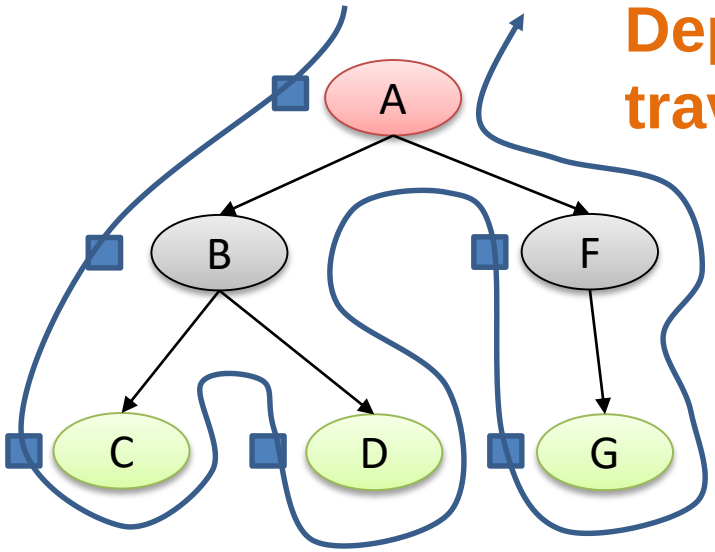
- Binary tree traversals
  - Depth-first traversals
    - Preorder traversal (NLR): root node first
    - Inorder traversals (LNR): Left subtree first
    - Postorder traversals (LRN): root node post left/right subtree



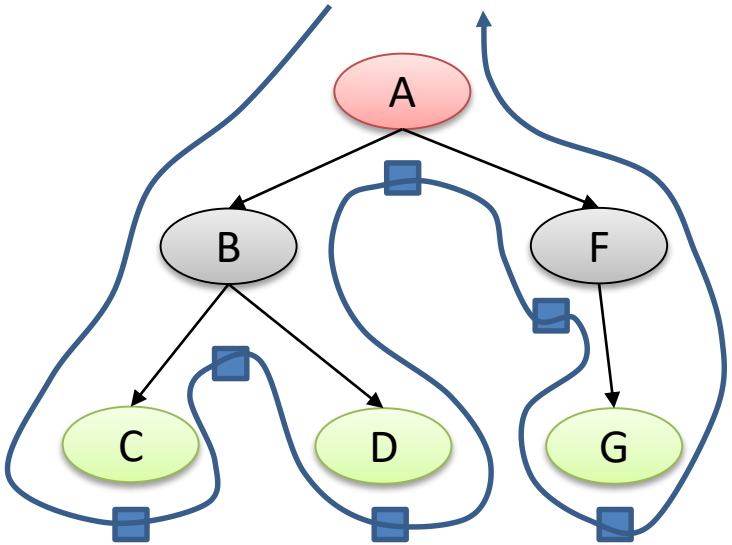
  - Breadth-first traversals

Binary trees: operations

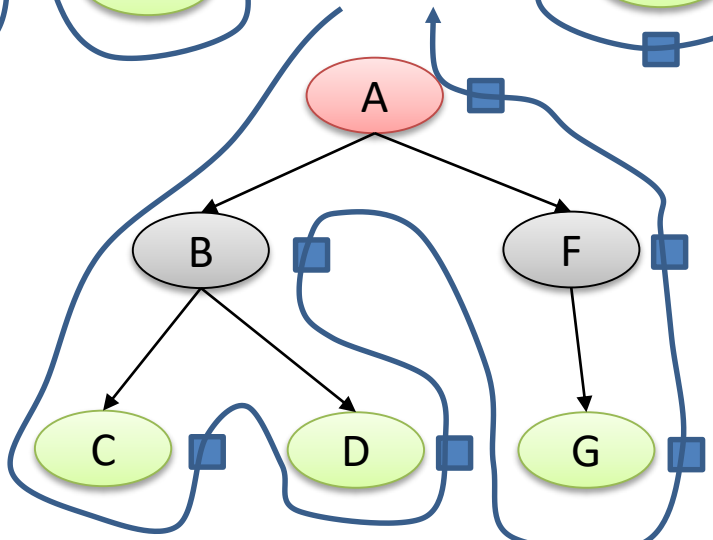
Depth-first traversals



NLR



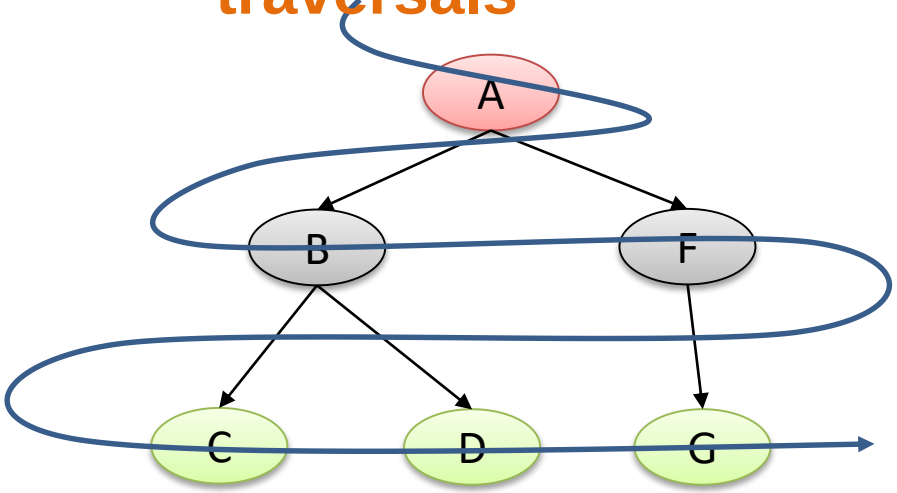
LNR



LRN

Binary trees: operations

Breadth-first traversals



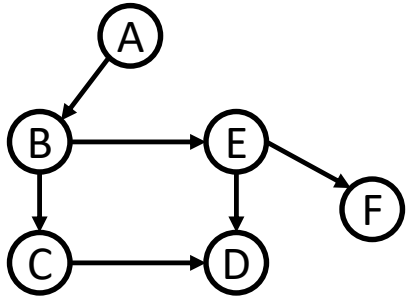
所有图片均来自网络

Trees

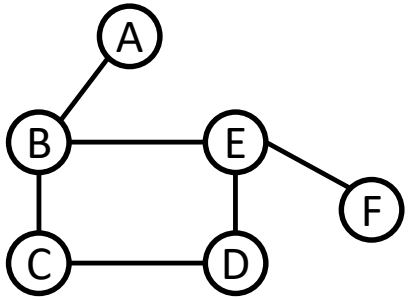
- Implementation: commonly by linked list!

Graphs

- A graph is a collections of nodes, called vertices, and a collections of line segments, called lines.
  - Directed graph
  - Undirected graph
- Two vertices in a graph are said to be **adjacent vertices** if an line directly connects them.



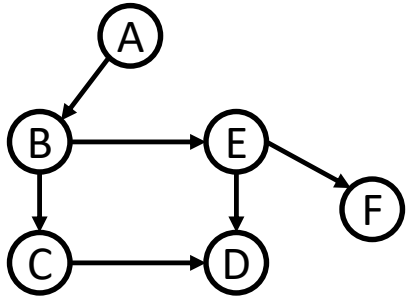
a. directed graph



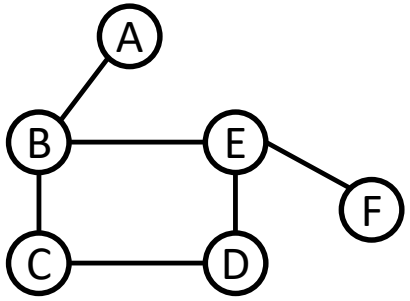
b. undirected graph

Graphs

- A path is a sequence of vertices in which each vertex is adjacent to the next node.
  - Cycle, loop
- Two vertices are said to be connected if there is a path between them
- A graph is said to be connected if, suppressing direction, there is a path from any vertex to any other vertex.
- A disjoint graph is not connected.



a. directed graph

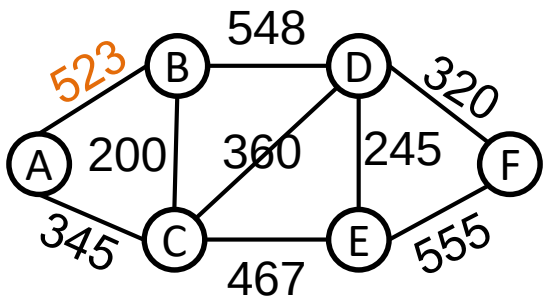


b. undirected graph

Graphs: operations

- Add/delete/find vertex
- Add/delete edge
- Traversal graph: process each vertex only once!
  - Depth-first
  - Breadth-first

Graphs: implementations



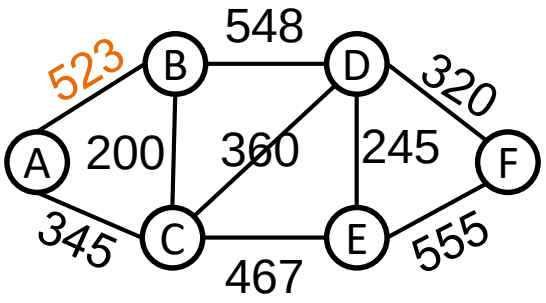
A
B
C
D
E
F

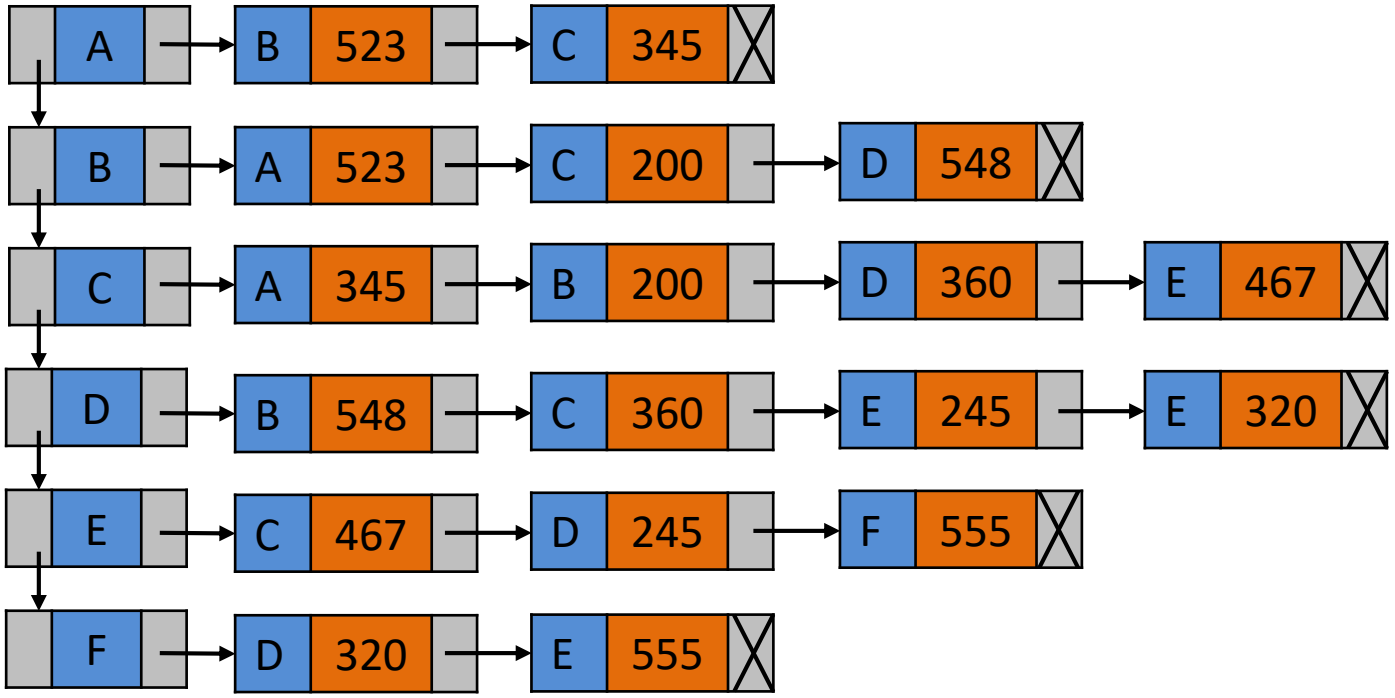
vertex array

	A	B	C	D	E	F
A	0	523	345	0	0	0
B	523	0	200	548	0	0
C	345	200	0	360	467	0
D	0	548	360	0	245	320
E	0	0	467	245	0	555
F	0	0	0	320	555	0

Adjacency matrix

Graphs: implementations





vertex list **adjacency list**