FISF130020: Introduction to Computer Science

Lecture 4: Data Structure

Hui Xu

xuh@fudan.edu.cn

Suppose the task is to
develop a trading system...

How to Describe a Company? *e.g.*, NVDIA

HOME > NVDA • NASDAQ

NVIDIA Corp

$116.00 ↓10.33% -13.37 1M

Sep 23, 12:31:01 AM UTC-4 · USD · NASDAQ · Disclaimer

1D 5D 1M 6M YTD 1Y 5Y MAX

Compare to

Tesla Inc	$238.25	TSLA	↑8.14%
Apple Inc	$228.20	AAPL	↑0.60%
Microsoft Corp	$435.27	MSFT	↑4.43%
Amazon.com Inc	$191.60	AMZN	↑8.22%

PREVIOUS CLOSE	$117.87
DAY RANGE	$115.39 - $118.62
YEAR RANGE	$39.23 - $140.76
MARKET CAP	2.85T USD
AVG VOLUME	332.34M
P/E RATIO	54.48
DIVIDEND YIELD	0.03%
PRIMARY EXCHANGE	NASDAQ

Data Structure of a Company

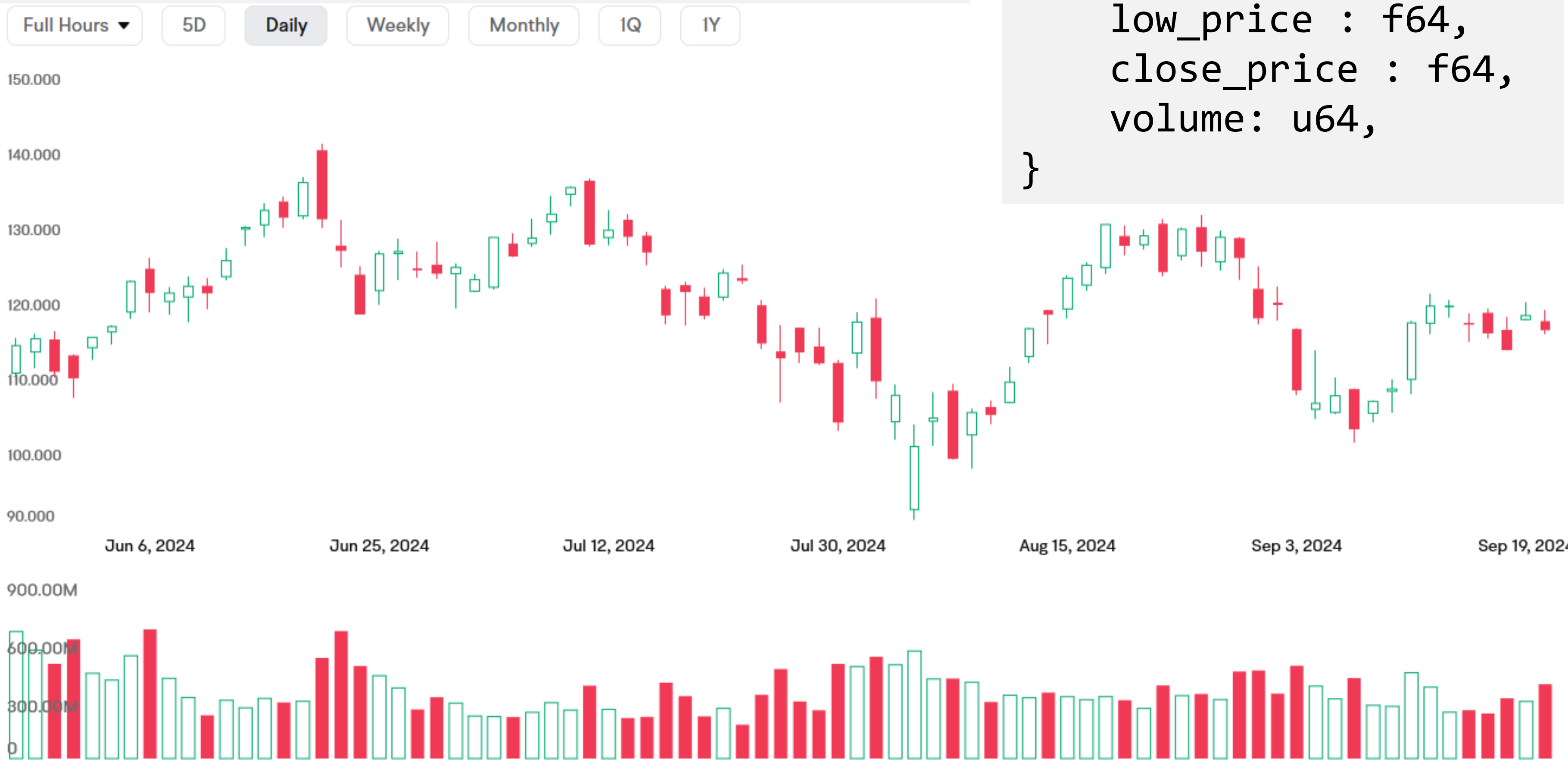
```
struct Company {
    name: String,                        // "NVIDIA Corporation"
    ticker: String,                      // "NVDA"
    exchange: String,                    // "NASDAQ"
    // trading info
    current_price: f64,                  // $100.1
    open_price: f64,                     // $99.12
    close_price: f64,
    high_price: f64,
    low_price: f64,                      // $98.79
    volume: u64,                         // 1000
    // more info
    pe: u64,                             // $200,000,000
    pb: f64,                             // $500.0 million
    dividend,
    market_cap: f64,                     // $1000 billion
    ...
}
```

Client

- We want to display the historical price, *e.g.,* as a candlestick chart

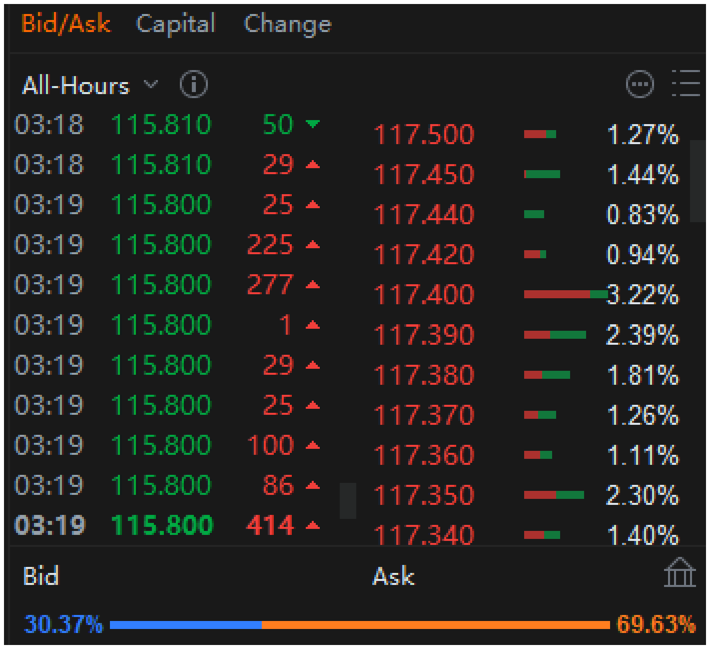
```
struct Company {
    historical_prices: Vec<TradingInfo>,
    ...
}
```

```
struct TradingInfo {
    date: U64,
    open_price: f64,
    high_price : f64,
    low_price : f64,
    close_price : f64,
    volume: u64,
}
```



Server

- How to manage the bid/ask orders?



Outline

1. Linear Data Structures

2. Trees and Graphs

3. In-class Practice

1. Linear Data Stuctures

Linear Data Stuctures

- Array
- List
- Queue
- Stack

Array

- A collection of elements stored in contiguous memory locations
- All elements are of the same data type
- Length: number of elements within the array
- Size: means the memory space it occupied

Memory Address	0x200	0x201	0x202	0x203	0x204	0x205	0x206	0x207	0x208	0x209
Data	**F**	**I**	**S**	**F**	**1**	**3**	**0**	**0**	**2**	**0**
Index	0	1	2	3	4	5	6	7	8	9

Array

- Access any elements via base address + offset
- Supposing the size of each data unit is 4 types, *e.g.*, 32bit integer, we can retrieve the *i*th data from the memory address:

```
a[i] = a[0] + 4*i
```

Memory Address	0x200	0x204	0x208	0x20c	0x210	0x214	0x218	0x21c	0x220	0x224
Data	**1**	**1**	**2**	**3**	**5**	**8**	**13**	**21**	**34**	**55**
Index	0	1	2	3	4	5	6	7	8	9

Array length: 10

Array size: 40 bytes

Array Operations

- Read/write any elements via base address + offset (constant time)
- Searching an element from an array of length n requires n/2 time
- Insertion or deletion an element requires shifting the rest elements

Memory Address	0x200	0x204	0x208	0x20c	0x210	0x214	0x218	0x21c	0x220	0x224
Data	**1**	**1**	**2**	**3**	**5**	**8**	**13**	**21**	**34**	**55**
Index	0	1	2	3	4	5	6	7	8	9

Two-dimensional Array: Matrix

- Consist of multiple one-dimensional array; each has the same length
- Supposing an i32 array has m rows, and each row has length n, we can retrieve data of the *i*th row and *j*th column:

```
a[i][j] = a[0][0] + 4*i*n + j
```

Row Index						
	0	**1**	**1**	**2**	**3**	**5**
	1	**8**	**13**	**21**	**34**	**55**
		0	1	2	3	4
		Column Index				

Array View

Memory Address	0x200	0x204	0x208	0x20c	0x210	0x214	0x218	0x21c	0x220	0x224
Data	**1**	**1**	**2**	**3**	**5**	**8**	**13**	**21**	**34**	**55**
Index	0, 0	0, 1	0, 2	0, 3	0, 4	1, 1	1, 2	1, 3	1, 4	1, 5

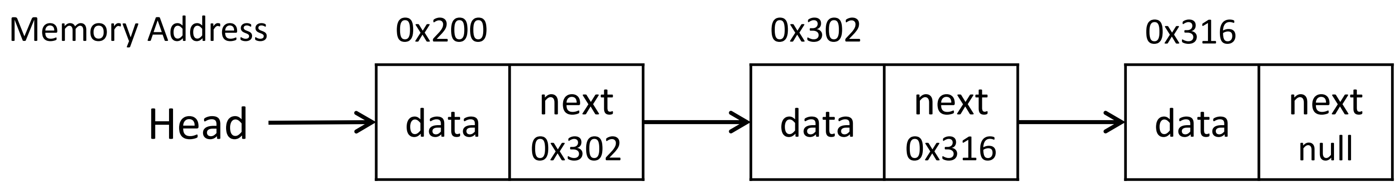
Memory View

Usage: Matrix Multiplication

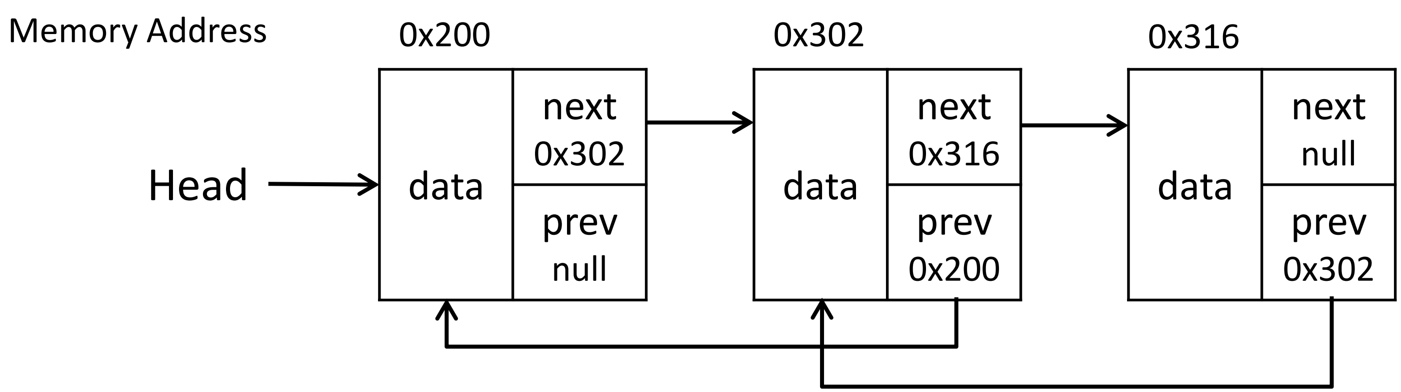
```
fn matrix_multiply(a: &Vec<Vec<i32>>, b: &Vec<Vec<i32>>) -> Vec<Vec<i32>> {
    let a_height = a.len();
    let b_height = b.len();
    let a_width = a[0].len();
    let b_width = b[0].len();
    if a_width != b_height {
         panic!("Matrix dimensions do not match for multiplication");
    }

    let mut result = vec![vec![0; b_width]; a_height];
    for i in 0..a_hight {
        for j in 0..b_width {
            for k in 0..a_width {
                result[i][j] += a[i][k] * b[k][j];
            }
        }
    }
    result
}
```

List

- Similar as array, but the data are not contiguous stored
- Each list node has a data field and an address field to the next or the previous node.



Single-linked List



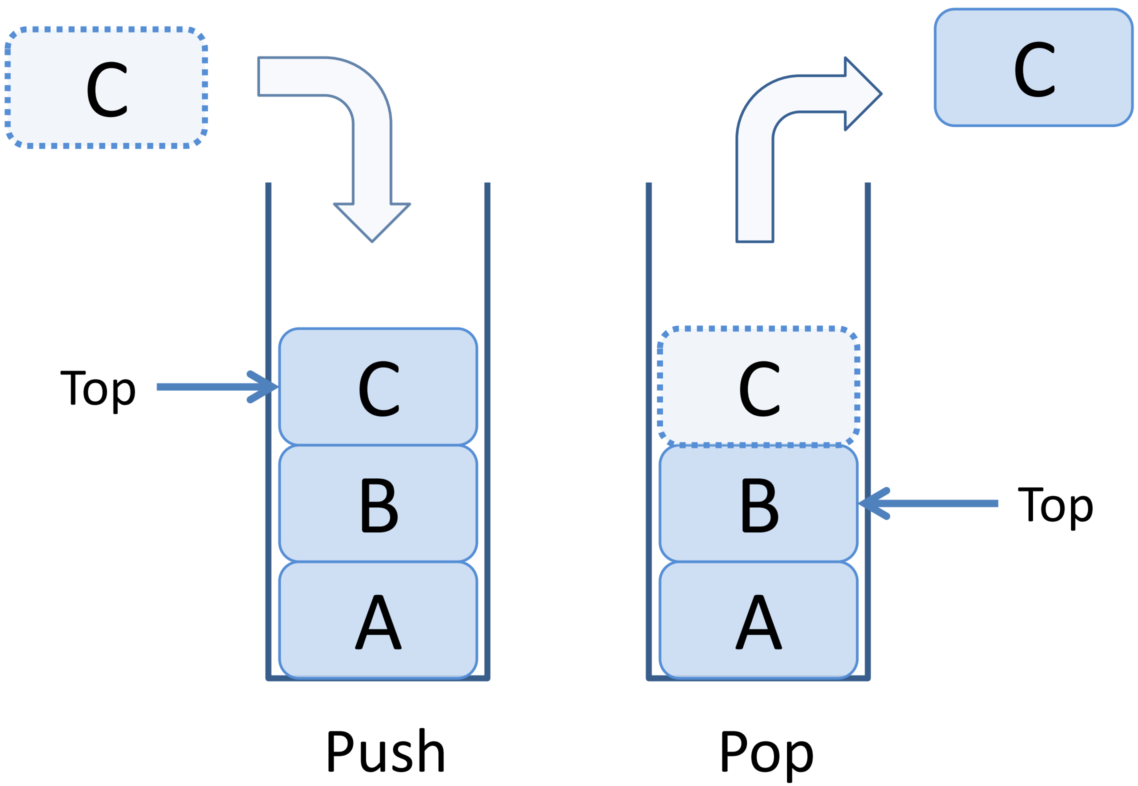
Double-linked List

Pros and Cons of List Operations

- The cost of accessing elements of different positions varies a lot
- Insertion/deletion an element at any positions costs constant time

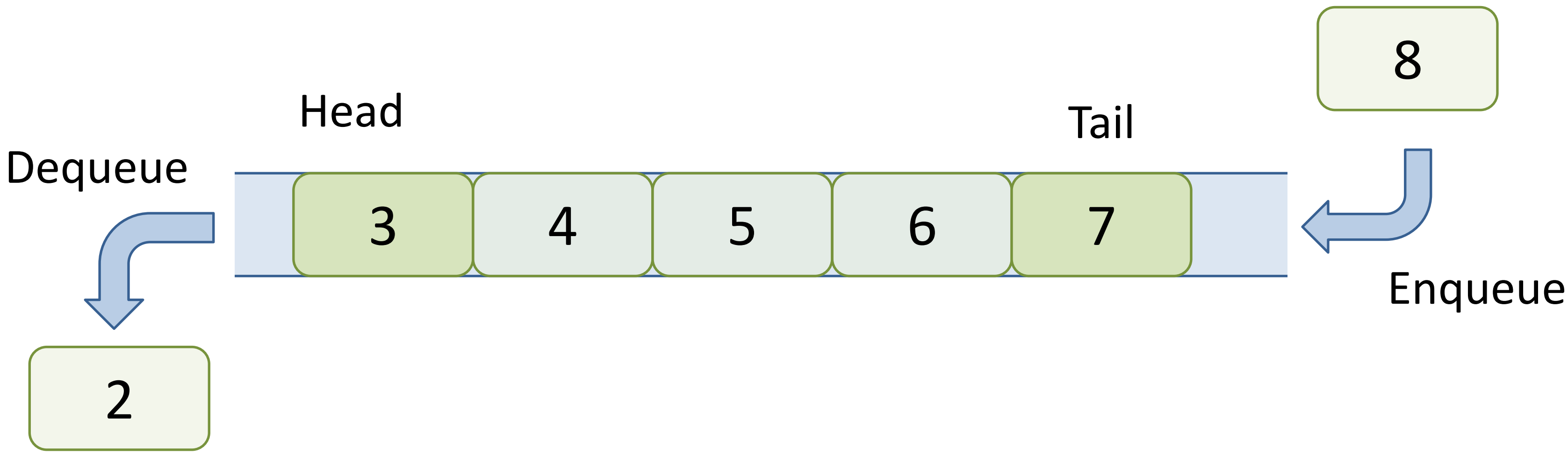
Stack

- A collection of elements with Last-In-First-Out (LIFO) order
- **push:** insert an element to the stack
- **pop:** remove the top element from the stack



Queue

- Similar as stack but with First-In-First-Out (FIFO) order
- Enqueue: add (or stores) an element to the end of the queue
- Dequeue: removal of an element from the queue



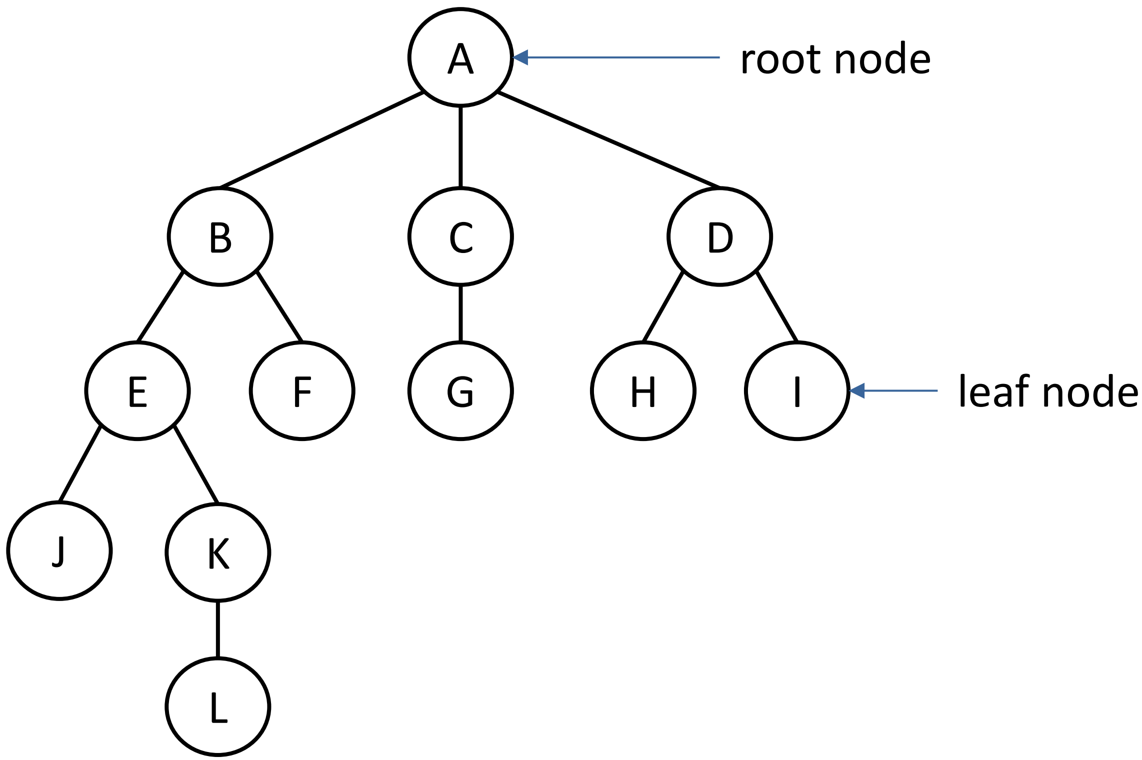
Question

- Which data structure do you recommend to use?
    - prices of a fixed range (one day)
    - prices of a dynamic range
    - ask/bid orders

2. Trees and Graphs

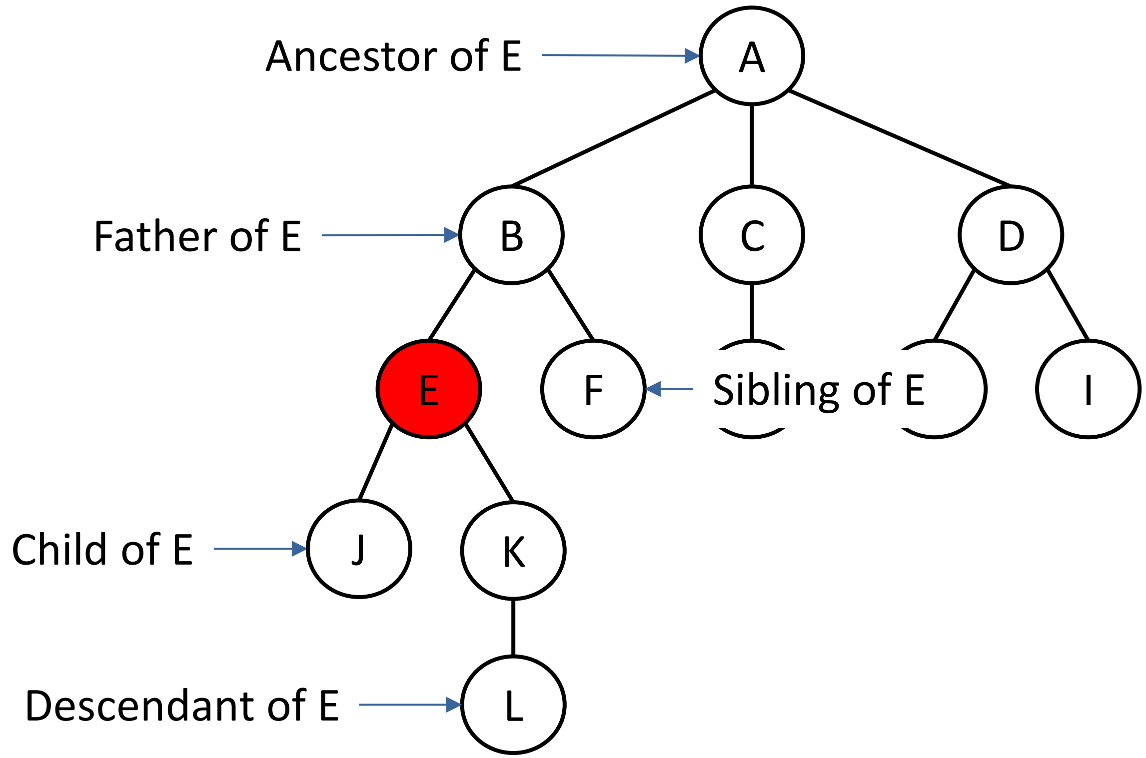
Trees

- Represent a hierarchical relationship among data units
- There is only one root node
- Each node may have one or more children excepect the leaf nodes



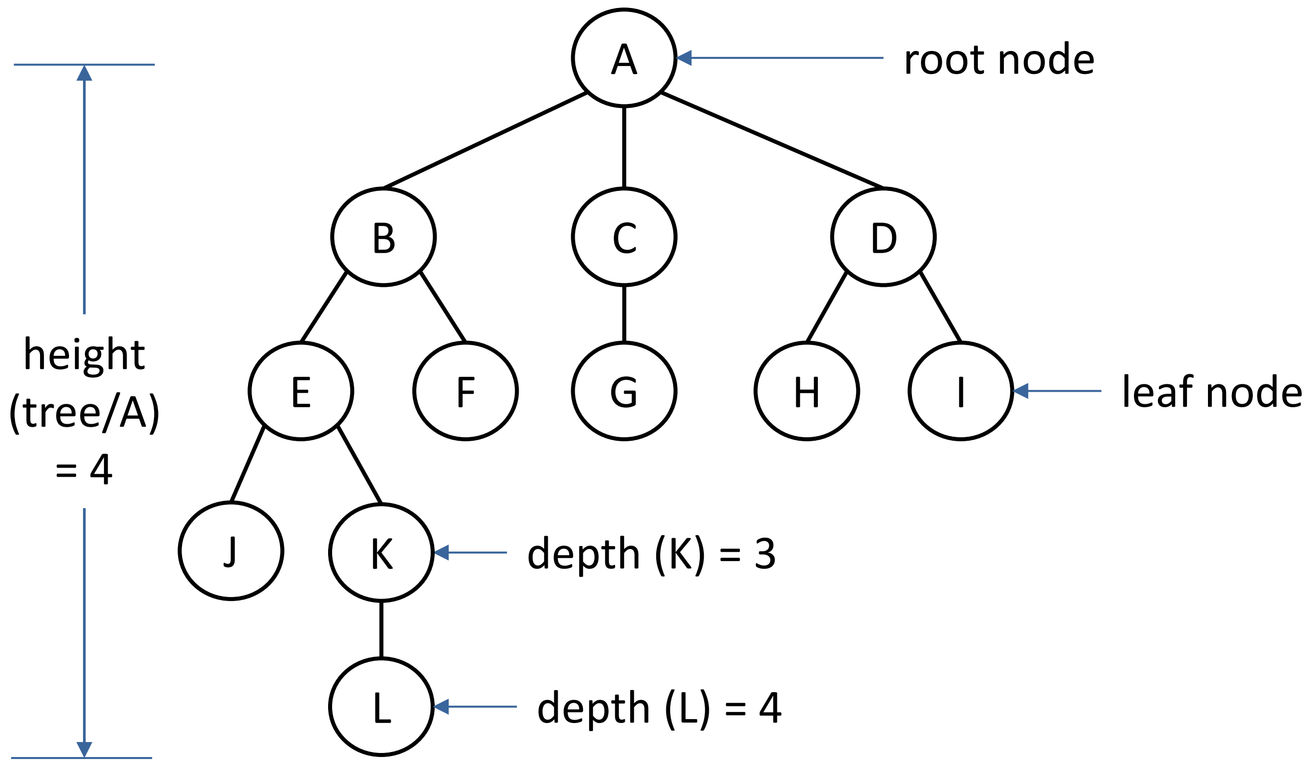
Relationships Among Nodes

- Ancestor
- Father node
- Sibling
- Child
- Descendant



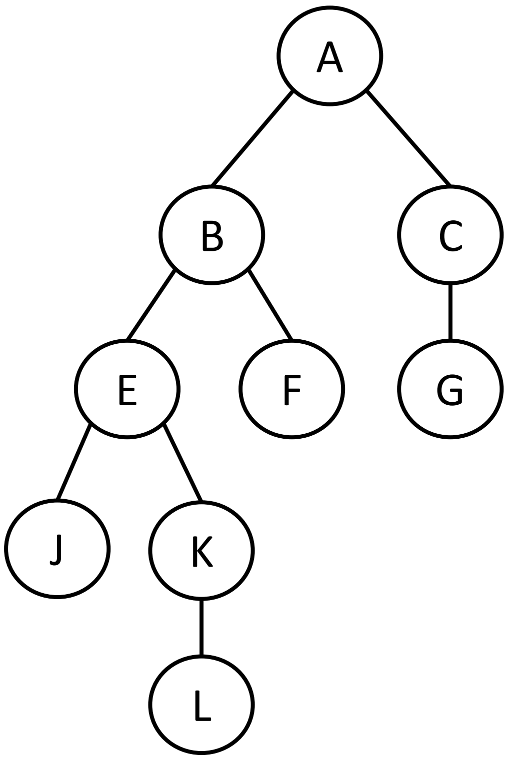
Terminologies

- Tree height: number of edges along the longest downward path from the root to a leaf
- Node depth: number of edges from the root to a node
- Degree of a node: number of children

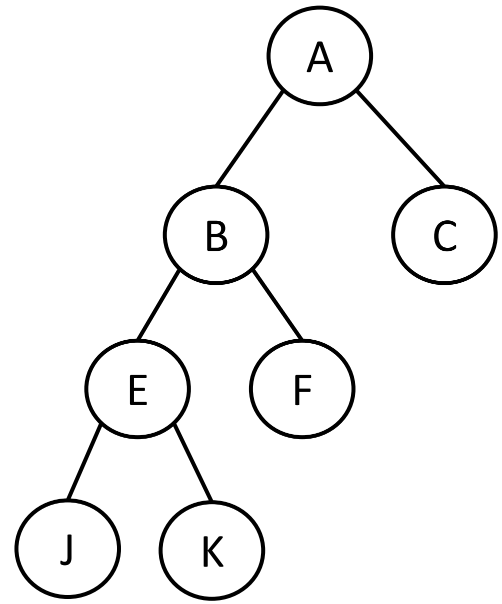


Binary Tree

- The degree of each node on a tree is at most two
- Full binary tree: the degree of each node on a tree is either 2 or 0



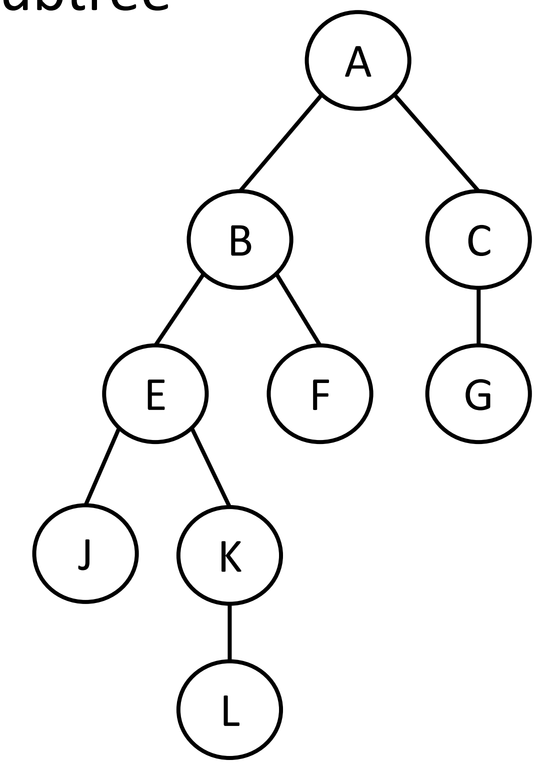
Binary Tree



Full Binary Tree

Tree Traversal

- Pre-order Traversal: root => left subtree => right subtree
  - A=>B=>E=>J=>K=>L=>F =>C=>G
- Post-order Traversal: left subtree => right subtree => root
  - J=>L=>K=>E=>F=>B=>G=>C=>A
- In-order Traversal: left subtree => root => right subtree
  - J=>E=>K=>L=>B=>F=>A=>C=>G

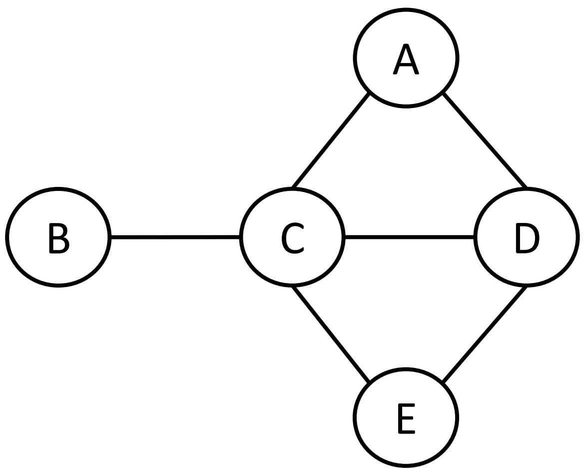


Question

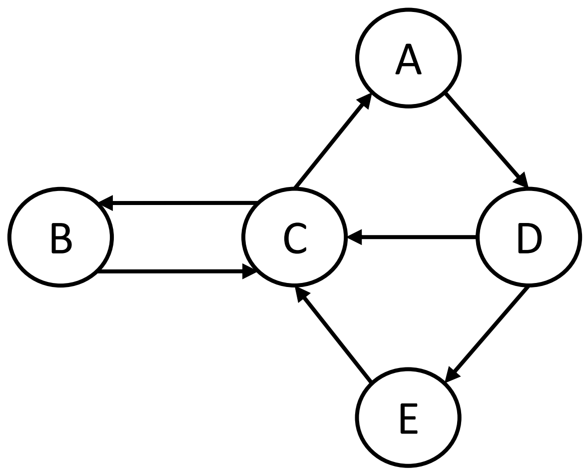
- How to define the data structure of a binary tree?
- How to define the data structure of a tree?

Graphs

- Similar to trees except that:
  - there is no partial order among nodes
  - there could be loops



undirected graph

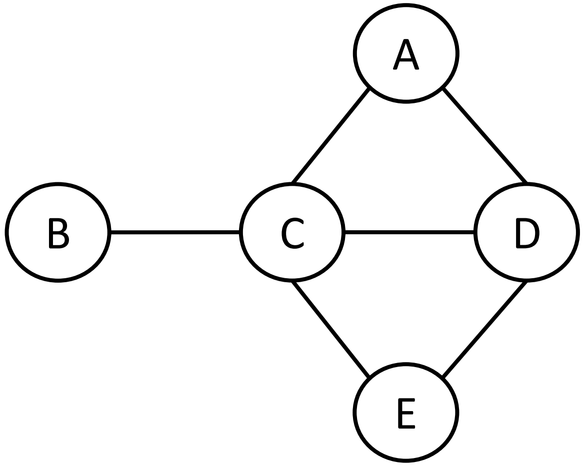


directed graph

Question

- How to define the data structure of a undirected graph?
- How to define the data structure of a directed graph?

Adjacent Matrix for Graph Representation



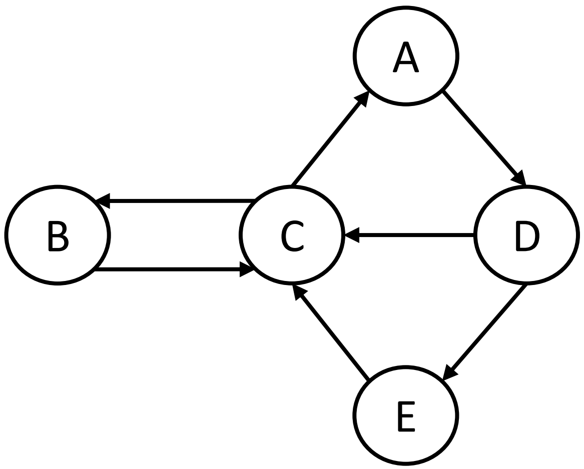
undirected graph

	A	B	C	D	E
A	0	0	1	1	0
B	0	0	1	0	0
C	1	1	0	1	1
D	1	0	1	0	1
E	0	0	1	1	0

The adjacency matrix is symmetric for undirected graphs

$$adj(i,j) = \begin{cases} 1, \text{if there is an edge between vertex } i \text{ and vertex } j \\ 0, \text{otherwise} \end{cases}$$

Adjacent Matrix for Graph Representation



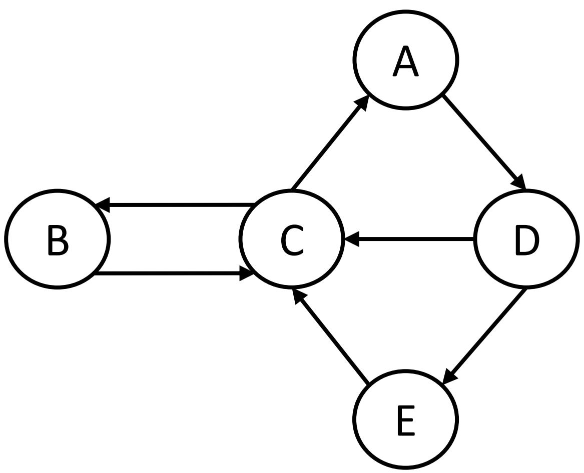
directed graph

	A	B	C	D	E
A	0	0	0	1	0
B	0	0	1	0	0
C	1	1	0	0	0
D	0	0	1	0	1
E	0	0	1	0	0

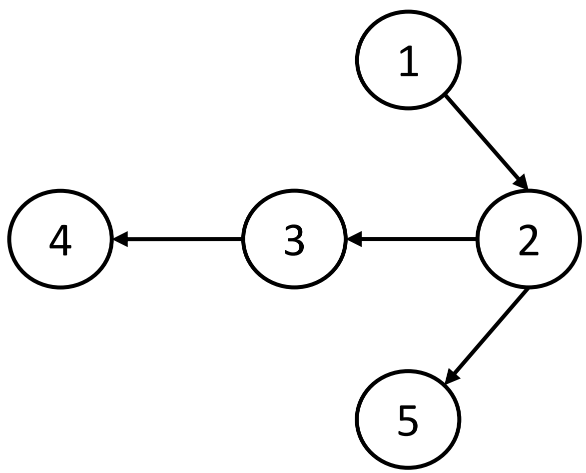
$$adj(i,j) = \begin{cases} 1, \text{if there is an edge from } i \text{ to } j \\ 0, \text{otherwise} \end{cases}$$

Graph Traversal

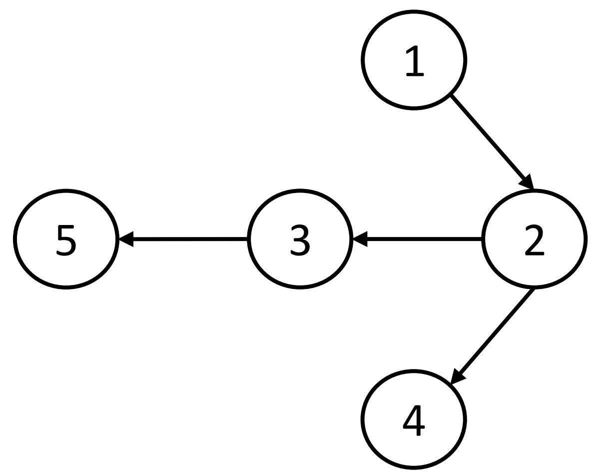
- Depth-first: explore as far as possible along each branch before backtracking
  - A=>D=>C=>B=>C(visited) => back to B=>back to C=>back to D=>E=>
  - C(visited)=>back to D=>back to A
- Width-first: explores all the nodes at the present depth level before moving on to nodes at the next depth level



directed graph



depth first



width first

3. In-class Practice

Option 1: Tree

- Implement a binary tree with Rust
- Traverse the tree and search a node with a particular value

Option 2: Trading Software (Client/Server-1)

- Design and implement a data structure to represent a set of companies
- Instantiate several companies with real data
- Implement the feature of querying a company based on the name, ticker symbol, or ID number

Option 3: Trading Software (Server-1)

- Design and implement a data structure for managing a set of orders
- Instantiate several orders with mock data
- Discuss the design considerations you took into account