CSE 431/531: Analysis of Algorithms (Summer 2023)
Graph representation and basic graph algorithms

Chen Xu

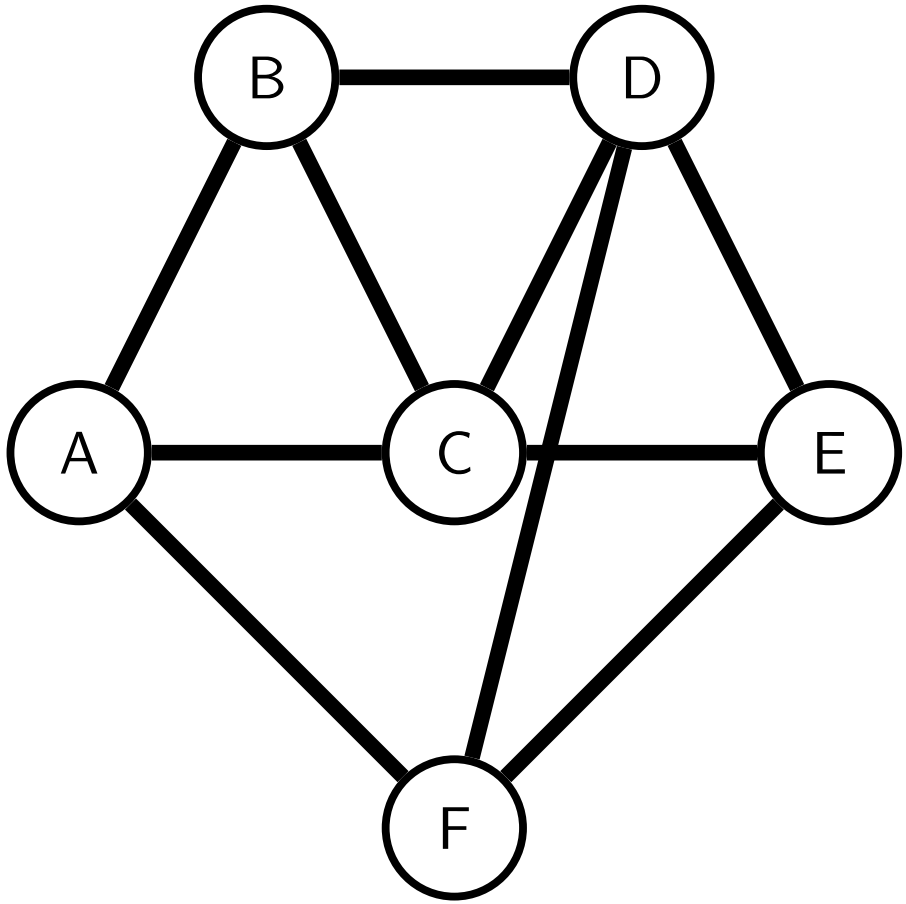
June 8, 2023

We will cover

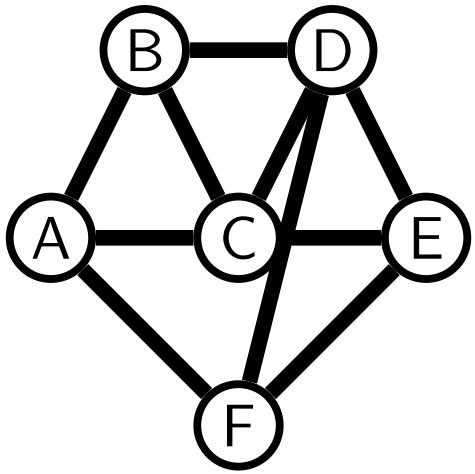
- Definitions of different graphs
- Basic graph algorithms

What is a graph?

- A **graph** is a collection of points, called **vertices**, and lines between those points, called **edges**.
- Graphs are used to represent networks of communication, data organization, computational devices, and the flow of computation, etc.

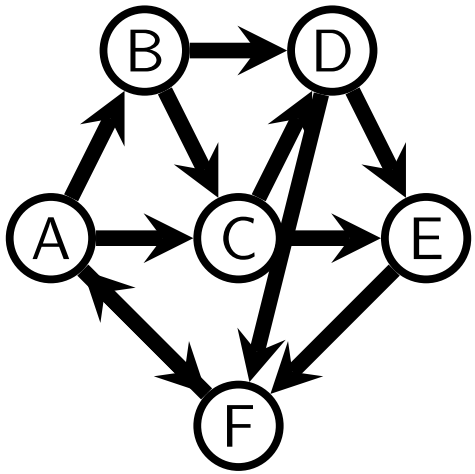


Undirected graph



- We use the set of **V** represent vertices and **E** represent edges. And write graph **G=(V,E)**.
- The set of vertices is $V = \{A, B, C, D, E, F\}$.
- The set of edges is $E = \{(A, B), (B, C), (C, D), (D, E), (E, F), (F, A), (A, C), (B, D), (C, E), (D, F)\}$, for undirected graph, $(A, B) = (B, A)$

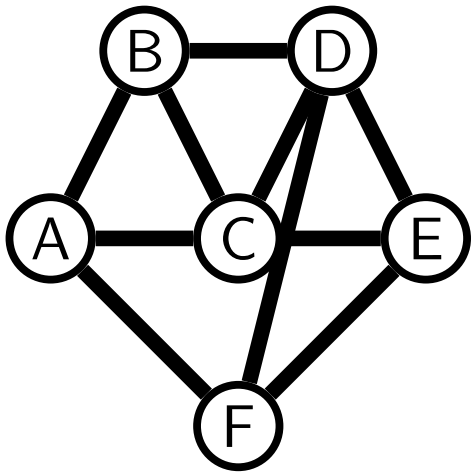
Directed Graph



- The set of vertices is $V = \{A, B, C, D, E, F\}$.
- The set of edges is $E = \{(A, B), (B, C), (C, D), (D, E), (E, F), (F, A), (A, F), (A, C), (B, D), (C, E), (D, F)\}$, for directed graph, (A, B) and (B, A) are different.

Matrix representation of undirected graphs

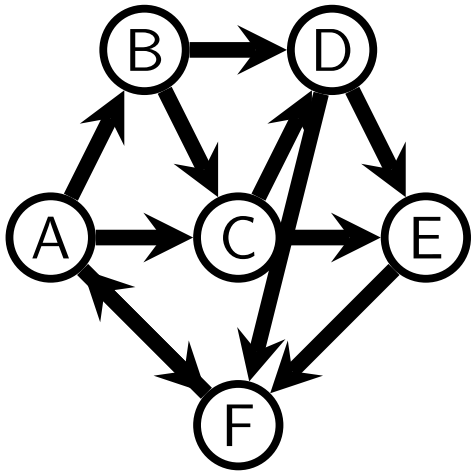
One way to represent a graph is using an *adjacency matrix*.



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

The adjacency matrices for undirected graphs are symmetric.

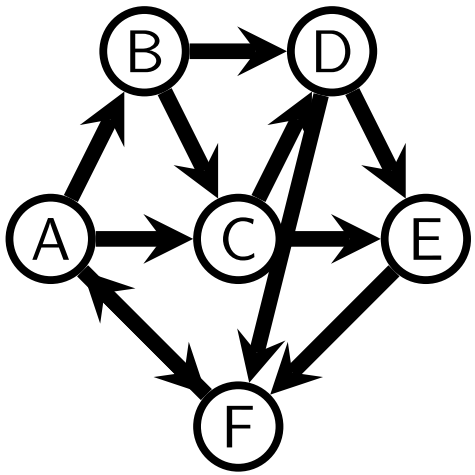
Matrix representation of directed graphs



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

The adjacency matrices for directed graphs are usually asymmetric.

Linkedlist representation of graphs



A: B → C → F

B: C → D

C: D → E

D: E → F

E: F

F: A

For undirected graph, we define the degree d of a vertex u as d_u. We can tell that $d_A = 3$ from the linked list.

For directed graph, we distinguish the degree for incoming edges as in-degree and outgoing edges as out-degree.

Comparison of the two representations

Table: Matrix vs Linked List

Criteria	**Matrix**	**Linked List**
Memory	$O(V^2)$	$O(V+E)$
Edge Lookup for (u, v)	$O(1)$	$O(d_u)$
Add Edge	$O(1)$	$O(1)$
List all neighbors of v	$O(n)$	$O(d_v)$
Iterate Over Edges	$O(V^2)$	$O(E)$

Connectivity

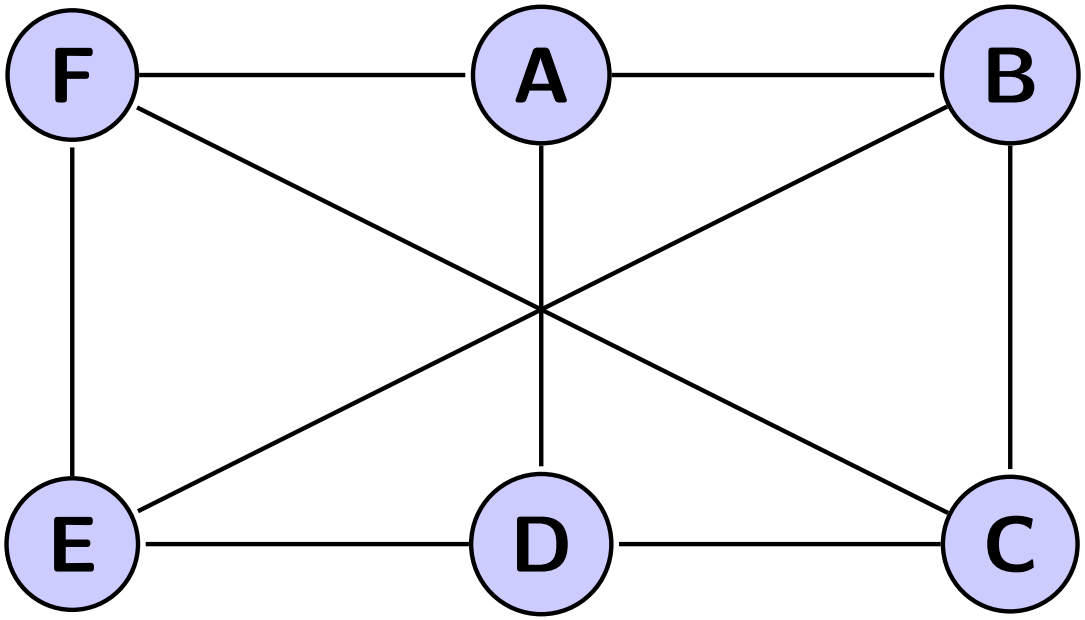
s-t Connectivity

Instance: $G = (V, E)$ and two vertices $s, t \in V$
Problem: Is there a path from s to t?

To represent a path P, we just need to include the sequence of edges. We say two edges are **incident** if they share the same vertex. We can also say an edge is **incident** to a vertex if the vertex is one of the endpoints.
To solve this problem, we introduce the concept of **Bread First Search(BFS)** and **Depth First Search(DFS)**.

BFS Example

Let's use BFS to traverse the following graph starting at vertex A:



After performing BFS starting from vertex A, the order of the visited vertices is: A, B, F, D, C, E.
BFS can be performed on directed graph too. We are using **Queue** as the data structure.

Using BFS to solve s-t Connectivity

BFS for s-t Connectivity

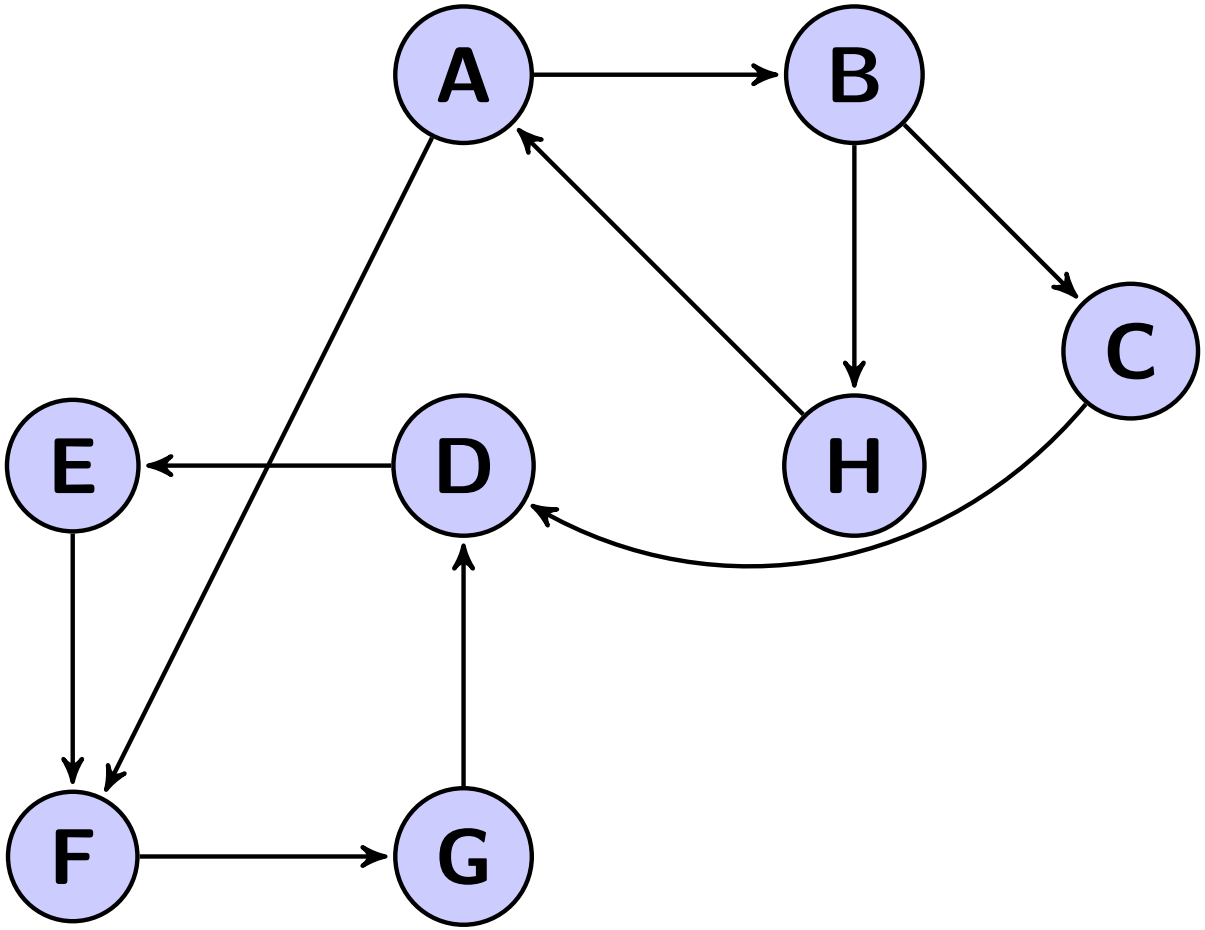
Input: G, s, t

Output: True/False if t is reachable from s.

```
1:  BFS_STCON(G, s, t)
2:      visited[] ← [False]
3:      Q ← []
4:      visited[s] ← True; Q.enqueue(s);
5:      while !Q.isEmpty() do
6:          v ← Q.dequeue()
7:          if v = t then return true                ▷ Target vertex reached
8:          for each neighbour u of v do
9:              if visited[u] = False then
10:                 visited[u] ← True; Q.enqueue(u);
11:     return false                    ▷ Target vertex not reachable from source
```

Running time is $O(|V| + |E|)$.

DFS Example



After performing DFS starting from vertex A, one possible order of the visited vertices is: A, B, C, D, E, F, G, H. We use **Stack** for DFS.
Exercise: What can be other orders?

Using DFS to solve s-t Connectivity

DFS for s-t Connectivity

```
1:  DFS_STCON(G, s, t)
2:      visited[] ← [False]
3:      S ← []
4:      S.push(s)
5:      while !S.isEmpty() do
6:          v ← S.pop()
7:          if v = t then return True                ▷ Target vertex reached
8:          for each neighbour u of v do
9:              if !visited[u] then
10:                 visited[u] ← True; S.push(u);
11:     return False                  ▷ Target vertex not reachable from source
```

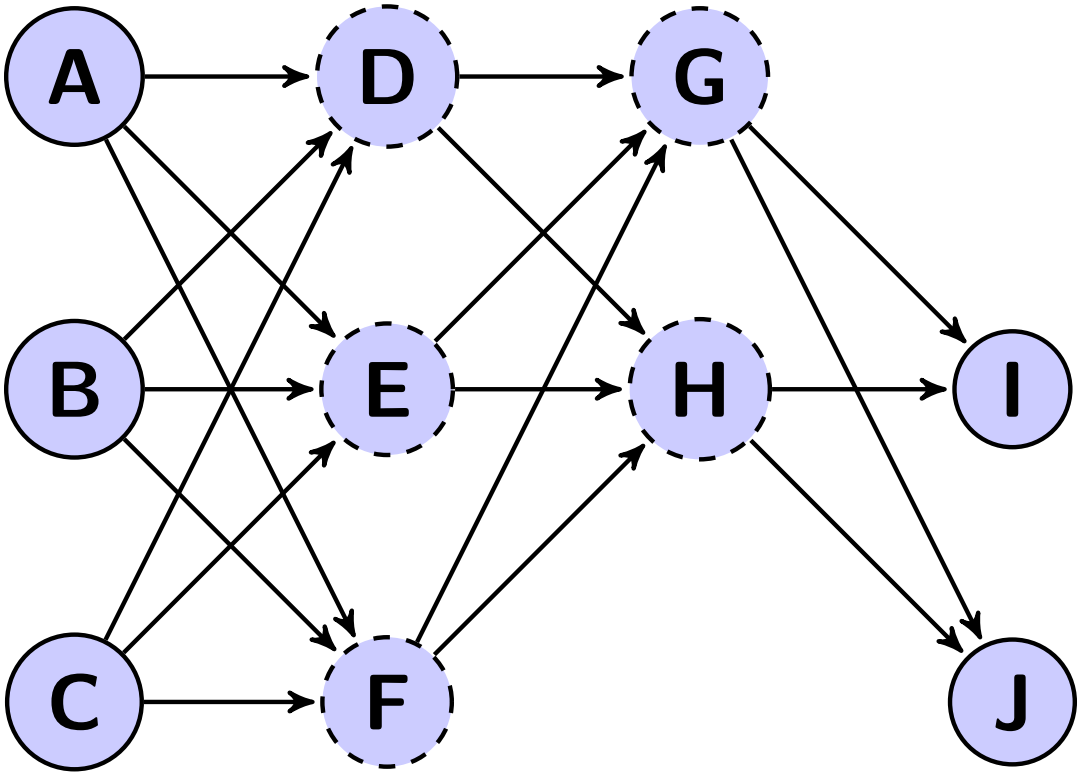
Running time is still $O(|V| + |E|)$

The first idea of solving graph problems

- BFS and DFS are the first two approaches to try on graph problems.
- In reality, the two methods can vary in average performance.
- Let us introduce DAG and then look at another example.

Directed Acyclic Graph - DAG

A Directed Acyclic Graph (DAG) is a directed graph with no directed cycles. That is, it consists of vertices and edges, with each edge directed from one vertex to another, such that following those directions will never form a closed loop.



DAGs have several important applications in computer science, such as scheduling, data compression, and data mining.

Cycle Detection Using DFS

DFS Cycle Detection

```
1:  HasCycle(G, v)
2:      visited[] ← [False]; inStack[] ← [False];
3:      S ← []
4:      visited[v] ← True; inStack[v] ← True;
5:      for each neighbour u of v do
6:          if !visited[u] then
7:              if HasCycle(G, u) then return True
8:              else if inStack[u] then return True
9:      S.remove(v); inStack[v] ← False;
10:     return False
```

BFS?

You may try implement BFS by yourself!