# CS 430 Lecture 24 Activities

Relationships between items:

$1 - 1$ List, Stack, Queue

$1 - N$ Tree, Hierarchical

$N - N$ Any graph item can be related to any other item.

## Opening Questions

1. Give an example NOT discussed in the video lecture of a problem that can be represented by a graph.

   The connections between airports.

2. If there is a path in a graph from a vertex back to itself, that is called a

   <u>cycle</u> [1].

3. Which representation of a graph, adjacency-list and adjacency-matrix, usually uses more memory and why?
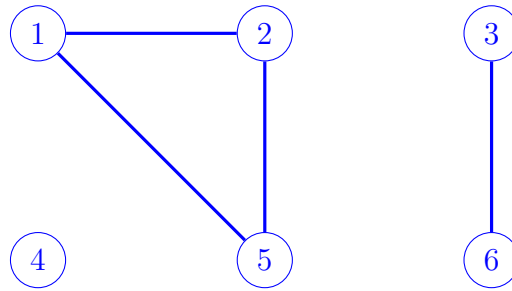
## Graphs

1. Draw the graph: A directed graph $G = (V, E)$ where $V = \{1, 2, 3, 4, 5, 6\}$ and $E = \{(1, 2), (2, 2), (2, 4), (2, 5), (4, 1), (4, 5), (5, 4), (6, 3)\}$. What is the edge $(2, 2)$ called?
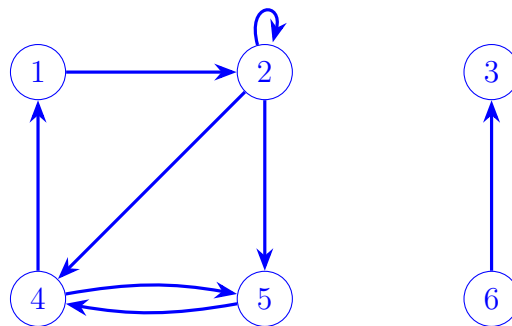


   The edge $(2, 2)$ is a self-loop.

2. Draw the graph: An undirected graph $G = (V, E)$ where $V = \{1, 2, 3, 4, 5, 6\}$ and $E = \{\{1, 2\}, \{1, 5\}, \{2, 5\}, \{3, 6\}\}$. What is vertex 4 called? What is the difference about how an edge set $E$ is denoted for an undirected graph? Are self-loops allowed in an undirected graph?

---

[1] A path of one edge is called a self-loop.

A vertex that is has no edges (is not connected to anything) is isolated.[2] The difference between how an edge set $E$ is denoted for an undirected graph versus a directed graph is that the directed graph is a tuple (where order matters) while an undirected graph is a set (where order doesn't matter). Self-loops are not allowed in an undirected graphs.

3. Define these terms:

   - Vertex $v$ is adjacent to vertex $u$ in an undirected graph.
     There is an edge $\{u, v\}$ (which is equivalent to $\{v, u\}$).
   - Vertex $v$ is adjacent to vertex $u$ in a directed graph.
     $(u, v) : u \rightarrow v$, which is not the same as $(v, u)$.
   - The degree of a vertex in an undirected graph.
     The number of edges touching the vertex.
   - The degree of a vertex in a directed graph.
     There are two types of degrees.

     **Out Degree** The number of edges leaving the vertex.
     **In Degree** The number of edges entering the vertex.

     The total degree is the sum of the In Degrees and Out Degrees.
   - A path in an undirected graph.
     Sequences of edges from a vertex to a vertex where the reuse of edges is okay.
   - A path in a directed graph.
     Sequence of edges in order of ordered pairs (following arrows).
   - The length of a path.
     The number of edges on the path.
   - $v$ is reachable from $u$.
     There is a path from $u$ to $v$.
   - A simple path.
     A path with no reused vertices OR edges.
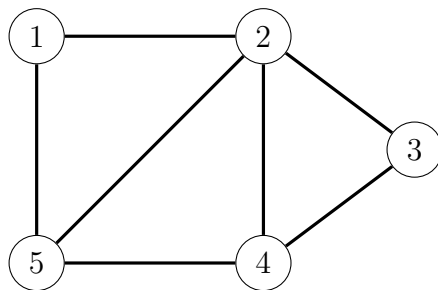   - A cycle in an undirected graph. What about a simple cycle?
     The path from a vertex to itself. A simple cycle is a cycle that doesn't reuse vertices or edges to get back to the original vertex (simple path back to the same vertex).

---

[2]In a directed graph, if the only connection a node has is to itself (a self-loop), then it is still considered isolated.

- A cycle in a directed graph. What about a simple cycle?
  The path in the ordered direction from a vertex to itself.

- Acyclic graph.
  No cycles in the graph.

- Connected undirected graph.
  Every vertex is reachable from every other vertex.

- Connected directed graph.
  Also known as a strongly connected graph. Every vertex is reachable (along edges in the correct direction) from every other vertex.

- Bipartite Graph.
  Separate the vertices into 2 sets. All edges go between the 2 sets. There are no edges between vertices in the same set.
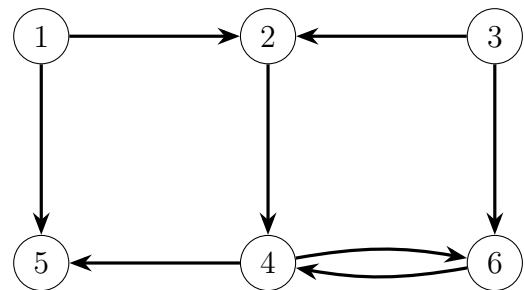
## Graph Implementations

4. What is the adjacency list implementation of these two graphs?



| 1 | $\rightarrow 5 \rightarrow 2$ |
| 2 | $\rightarrow 1 \rightarrow 5 \rightarrow 4$ |
| 3 | $\rightarrow 2 \rightarrow 4$ |
| 4 | $\rightarrow 2 \rightarrow 3 \rightarrow 5$ |
| 5 | $\rightarrow 1 \rightarrow 2 \rightarrow 4$ |

No order to the list of vertices. The arrows are representing a linked list of who the vertex is connected to (NOT a set of edges.)

| 1 | $\rightarrow 2 \rightarrow 5$ |
| 2 | $\rightarrow 4$ |
| 3 | $\rightarrow 2 \rightarrow 6$ |
| 4 | $\rightarrow 5 \rightarrow 6$ |
| 5 | |
| 6 | $\rightarrow 4$ |

Memory: $O(|V| + |E|)$. Find if edge exists: $O(|V|)$

5. What is the adjacency matrix implementation of the above two graphs?

Memory: $O(|V|^2)$

|   | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 |
| 2 | 1 | 0 | 1 | 1 | 1 |
| 3 | 0 | 1 | 0 | 1 | 1 |
| 4 | 0 | 1 | 1 | 0 | 1 |
| 5 | 1 | 1 | 0 | 1 | 0 |

|   | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 | 0 | 1 |
| 4 | 0 | 0 | 0 | 0 | 1 | 1 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 1 | 0 | 0 |

6. How do the two implementations handle a weighted graph?

   Edges have numbers, add weights to the adj list and adj matrix.

7. Two different representations of the graph data structure are discussed in the book, adjacency-list and adjacency-matrix. Please briefly discuss the runtime (in terms of $|V|$ and $|E|$ of these graph operations/algorithms using each implementation.) Assume vertices are labeled as integers.

   - What is the worst-case big-O runtime for checking to see if an edge from vertex $u$ to vertex $v$ exists?

     Adj list: $O(V)$, Adj matrix: $O(1)$

   - How long does it take to compute the out-degree of every vertex of a directed graph?

   - How long does it take to compute the in-degree of every vertex of a directed graph?

## Graph Traversals

A way to search/visit all the vertices in a graph. There is not a unique answer usually.

   - Undirected graph - if connected, all vertices will be visited.

   - Directed graph - Must be strongly connected to be able to visit all vertices.

Breadth first - visit vertices one edge from a given (or random) source, two edges from source, etc. Uses a queue and some way to make a vertex (white initially, gray when first visited and put in queue, black when out of queue), label a vertex how far from the source, and label a vertex with how its predecessor vertex was during the traversal.

8. Perform a breadth first search on this graph.

---
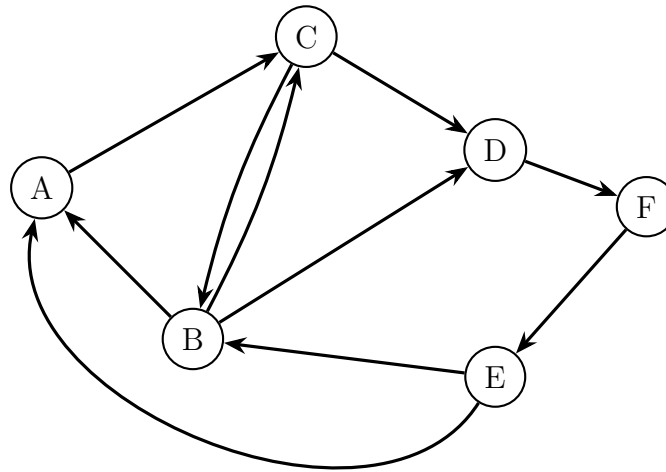
**Algorithm 24.1** Breadth-First Search for Graphs

---

1: **function** BFS($G$, $s$)
2:     **for all** vertex $u \in V[G] - \{s\}$ **do**
3:         COLOR($u$) ← WHITE                                    ▷ Unvisited
4:         $d[u] \leftarrow \infty$
5:         $\pi[u] \leftarrow$ NIL
6:     **end for**
7:     COLOR($s$) ← GRAY                              ▷ First time seen, put in queue
8:     $d[s] \leftarrow 0$                            ▷ $d$ is the distance from the start
9:     $\pi[s] \leftarrow$ NIL                                   ▷ $\pi$ is the predecessor
10:     $Q \leftarrow \emptyset$
11:     ENQUEUE($Q$, $s$)
12:     **while** $Q \neq \emptyset$ **do**
13:         $u \leftarrow$ DEQUEUE($Q$)
14:         **for all** $v \in$ ADJ($u$) **do**
15:             **if** COLOR($v$) == WHITE **then**
16:                 COLOR($v$) ← GRAY
17:                 $d[v] \leftarrow d[u] + 1$
18:                 $\pi[v] \leftarrow u$
19:                 ENQUEUE($Q$, $v$)
20:             **end if**
21:         **end for**
22:         COLOR($u$) ← BLACK                          ▷ Last time seen, out of queue
23:     **end while**
24: **end function**

---