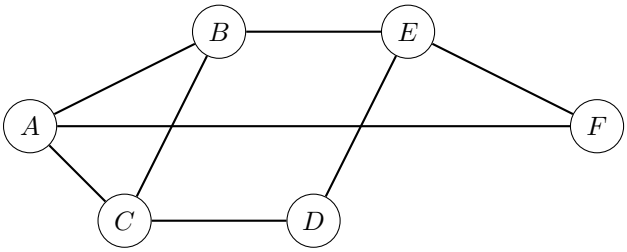
An undirected graph is specified by $G = (V, E)$. V denotes the set of vertices (sometimes called nodes). E denotes the edges between pairs of nodes. By convention, $n = |V|, m = |E|$.

A **self-loop** is an edge to and from the same node.

A **Simple graph** is a graph with no self-loops and no parallel edges.

The **degree** of a node in an undirected graph is the number of edges connected to it. A self-loop counts twice towards the degree of its *incident* vertex.

Nodes u, v are **neighbors**, or **adjacent** if they are connected by an edge. Given any set of nodes $A, N(A)$ is the set of nodes which are neighbors to a node in A.



Exercise 1. In the above graph, how many neighbors does B have? Name them.

Exercise 2. You have a graph with 10 vertices, and each node has degree 6. How many edges are there?

Exercise 3. Can you draw a graph with 5 vertices, each with degree 1?

Handshaking Theorem: Given an undirected graph with m edges, $2m = \sum_{v \in V} \deg(v)$.

Corollary to the Handshaking Theorem: An undirected graph has an even number of vertices with odd degree.

1 Representations of graphs

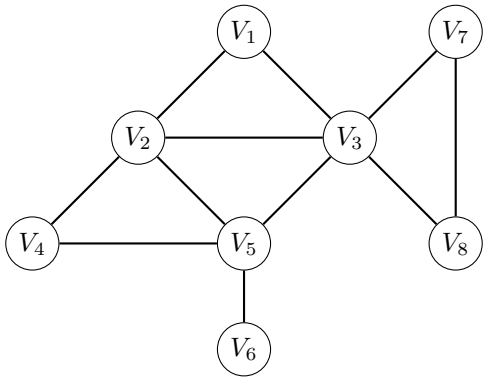
Two common ways to represent a graph are *adjacency list* and *adjacency matrix*. How would the graph above be represented in each format?

2 Graph Traversals

2.1 Depth-First Search

Given a graph G and two nodes s and t, how could we find a path from s to t?

For example, how do we find a path from V_1 to V_6 in the following graph?



Depth-first search creates a tree, rooted at the starting vertex. Because the path between two vertices in a tree is unique, we can say that the path found is the unique path from the root to any given vertex. We can express the code iteratively or recursively:

```
DFS-recursive(u)
  Mark u as "discovered"
  for each edge (u, v) do
    if v is not marked "discovered" then
      DFS-recursive(v)
    end if
  end for
```

```
DFS-iterative(s)
  ∀_v discovered[v] = false
  Initialize S to be a stack with s as its only element
  while S is not empty do
    u ← pop(S)
    if discovered[u] = false then
      discovered[u] = true
      for all edges (u, v) do
        push(S, v)
      end for
    end if
  end while
```

Question 1. We can see that depth-first search finds *some path* between v_1 and v_6; does this find the *shortest path*?

2.2 Breadth-First Search

Like DFS, this algorithm will create a tree rooted at the start vertex.

```
BFS(G, s)
  Set discovered[s] = true and discovered[v] = false for all other v
  L[0] ← {s}
  i ← 0
  while L[i] is not empty do
    Make L[i + 1] as empty list
    for all vertices u ∈ L[i] do
      for all edges (u, v) do
        if discovered[v] = false then
          discovered[v] ← true
          Add v to list L[i + 1]
        end if
      end for
    end for
    i ← i + 1
  end while
```

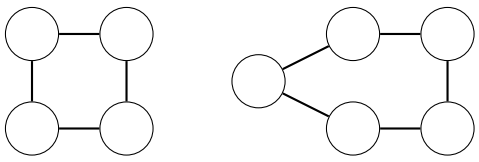
L_i consists of all nodes whose shortest path to s is length exactly i. If $(x, y) \in E$, then x, y differ by at most one level.

Question 2. Does Breadth-First Search find the shortest path between the start vertex and any other vertex? Do you have to make any assumptions about the edges' relative importance to say this?

2.3 Graph Coloring

The **graph coloring** problem is as follows. We are given a graph $G = (V, E)$ and a positive integer k. We want to find out if we can assign each vertex one of k colors in such a way that no edge has both of its endpoints the same color. Sometimes, we are not given the value k, and instead we want to find the smallest value of k for which the answer is "yes."

Question 3. A graph for which $k = 2$ is possible is known as "bipartite." Which of the following two graphs is bipartite?



C_4 C_5

Question 4. How can you use BFS to determine if a graph is bipartite?