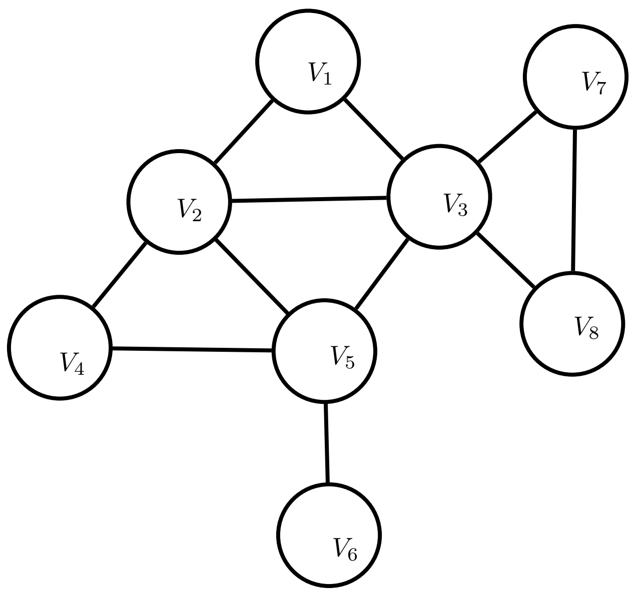
Graphs Fundamentals

Associated reading: K/T textbook, Ch. 3, G/T textbook, Ch. 13, Rosen textbook, Ch. 10



Definitions

- A *graph* G is a data structure composed of a set of vertices V and a set of edges E. Each edge connects two vertices, called its *endpoints*. In CSCI 570, we will deal only with graphs where each edge connects *exactly two distinct vertices* and where the set of vertices is finite.
- The following abbreviations are common when analyzing graphs: $|V| = n$ and $|E| = m$
- A *vertex* is typically drawn as a circle, above. The plural of vertex is *vertices*.
- A *path* P is a sequence of k vertices $v_1v_2 \dots v_{k-1}v_k$ such that for each pair $(v_i, v_{i+1}) \in E$.
- A path is *simple* if all vertices are distinct.
- A cycle is a path $v_1v_2 \dots v_{k-1}v_k$ such that $v_1 = v_k$, $k > 2$, and the first $k-1$ are all distinct.
- An undirected graph is *connected* if, for each pair of vertices u, v, there is a path (u, v).
- The *distance* between u and v is the minimum number of edges in a $u - v$ path.
- A *tree* is a connected acyclic graph.

  If G is an undirected graph on n vertices, any two of the following imply the third:

  - G is connected.
  - G does not contain a cycle.
  - G has $n - 1$ edges.

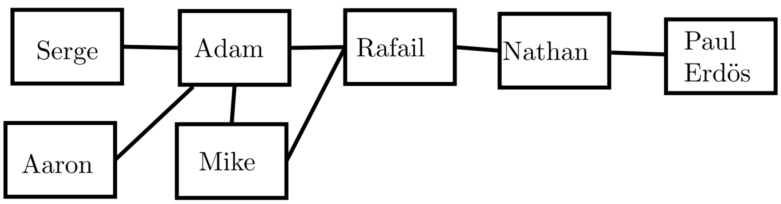
Some Types of Graphs

Question 1. You have a graph with 10 vertices, and each node has degree 6. How many edges are there?

Question 2. Can you draw a graph with 5 vertices, each with degree 1?

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

Collaboration Graphs have edges between any two people who have collaborated on a research paper. There was an extremely prolific mathematician named Paul Erdös, and people like to track their "Erdös Number", which is the length of the shortest path in a collaboration graph between them and Paul Erdös.



Traversing a Graph

Breadth-First Traversals

Let's do a breadth-first traversal from V_1 in the graph on the first page.

- The layer to which each vertex belongs corresponds to its distance from the starting vertex.
- Note that everything reachable *from s* is in the breadth-first search tree somewhere

Implementing Breadth-First Search

```
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]
    i ← i + 1
```

Question 3. What are two common ways to represent a graph in a computer program? What are the advantages and disadvantages of each?

Bipartite Graphs

- A graph is bipartite if it can be partitioned into two sets A and B such that, for all edges $e = (u, v) \in E$, u and v are in opposite sets.
- Alternate Definition:

Question 4. Suppose a graph is bipartite: how can I offer proof to you that it is?

Question 5. Suppose I offer that proof. How can you check that my proof is valid?

Question 6. What if the graph *isn't* bipartite? How would I convince you?

Question 7. True or False: Every tree is bipartite.

Question 8. True or False: Every graph with a cycle in it is *not* bipartite.

Question 9. How can we check if a graph is bipartite?

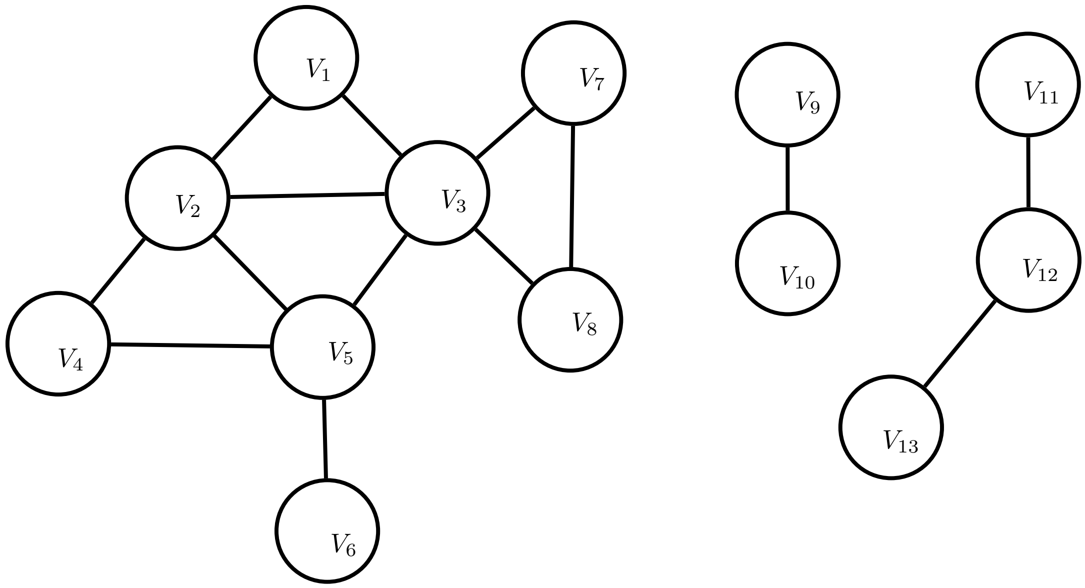
Depth-First Search

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

```
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)
```

- Suppose I do a BFS and a DFS, separately, on the same graph starting at the same vertex.
  - Do I visit the same vertices in both searches?
  - For the vertices that are visited in both searches, are they visited in the same order?
- How long does a depth-first search take?

Application: Listing all connected components



Initialize the list of discovered to all false, and remove the initialization line in Search
for all vertices v **do**
  if v unexplored **then**
    Search(v)

Directed Graphs

So far, edges have been symmetric; if there is an edge $e = (u, v)$, then there is also an edge $e = (v, u)$. In a *directed graph*, the edges are each one-way.

- What can we represent with a directed graph that we can't represent with an undirected one?
- What does it mean for a directed graph, or a component of one, to be "connected"?
  - What if the same definition of connected holds? $\forall_{u,v}$ $\exists$ path $u \to v$ and $\exists$ path $v \to u$.
  - What if that doesn't hold, but it would hold if the edges' directions were removed?

Directed Acyclic Graphs and Topological Ordering

Question 10. What type of problem might we represent with a directed graph such that it would have no cycles?

Question 11. We define a *topological order* in a DAG as an ordering $v_1, v_2, \ldots, v_n$ such that if v_i appears earlier in the order than v_j, there is no path in G from v_j to v_i.

- Does every DAG have a topological order?
- Is it the case that every graph with a topological order is a DAG?

`Topological-Sort idea:`

while G has vertices remaining **do**
  Select a vertex v with no incoming edges
  Output v
  Remove v (and its outgoing edges) from G

`Topological-Sort(G)`

Compute `incoming`[v] for each vertex
Create B, an empty bag data structure
$B \leftarrow$ all v with `incoming`[v] $= 0$.
while $B \neq \emptyset$ **do**
  Remove v from B
  Output v
  for each $w \in \text{adj}[v]$ **do**
    subtract one from `incoming`[w]
    if incoming[w] is now zero **then**
      add w to B