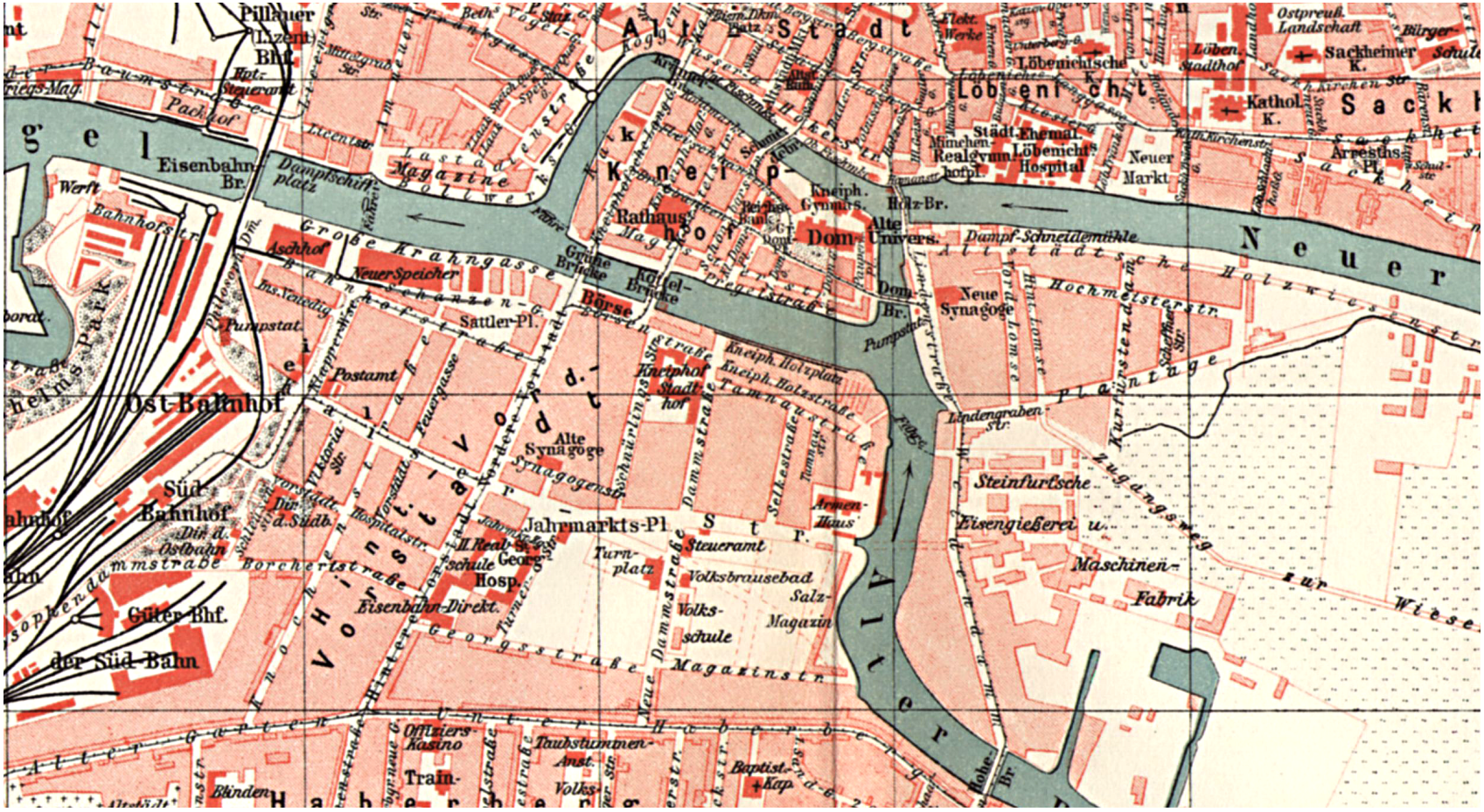
Graph Algorithms

The Seven Bridges of Königsberg

- Consider the seven bridges in Königsberg, Prussia: can one walk across every bridge exactly once?



Graphs

Definition

A **graph** $G = (V, E)$ consists of a set of vertices, V, and a set of edges, E.

Definition

An **edge** $e = (u, v)$ connects one or two **endpoints**, u and v, where $e \in E$ and $u, v \in V$.

- By convention, $|V| = n$ and $|E| = m$.
  - Some graphs are "dense", where $|E|$ dominates $|V|$.
  - Some graphs are "sparse", where $|E|$ does not dominate $|V|$.
- Graphs are used to model relationships between objects:
  - Define what each vertex represents.
  - Define when two vertices are connected by an edge.

Graphs

Example

Graphs can model routes between cities:

- Vertices represent cities.
- Two vertices are connected by an edge $e = (u, v)$ if and only if there is a highway directly between city u and city v.

Example

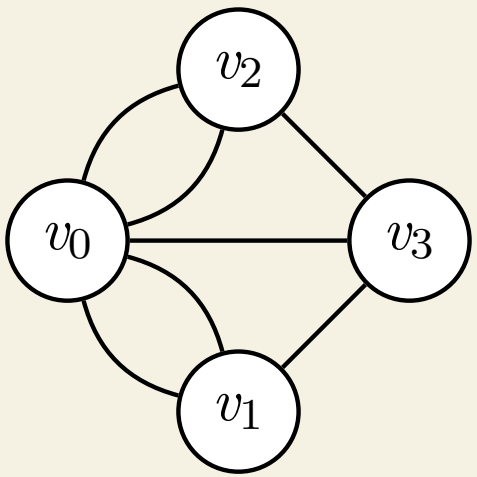
Graphs can model human populations:

- Vertices represent people.
- Two vertices are connected by an edge $e = (u, v)$ if and only if person u and person v make regular face-to-face contact.

Graphs

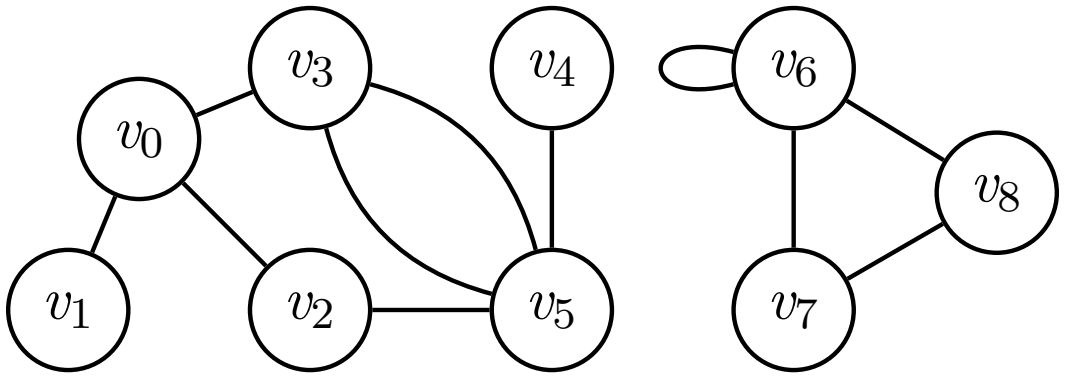
Example

Graphs can model the Seven Bridges of Königsberg:



- Vertices represent landmasses.
- Two vertices are connected by an edge $e = (u, v)$ if and only if there is a bridge between landmass u and landmass v.

Graph Terminology



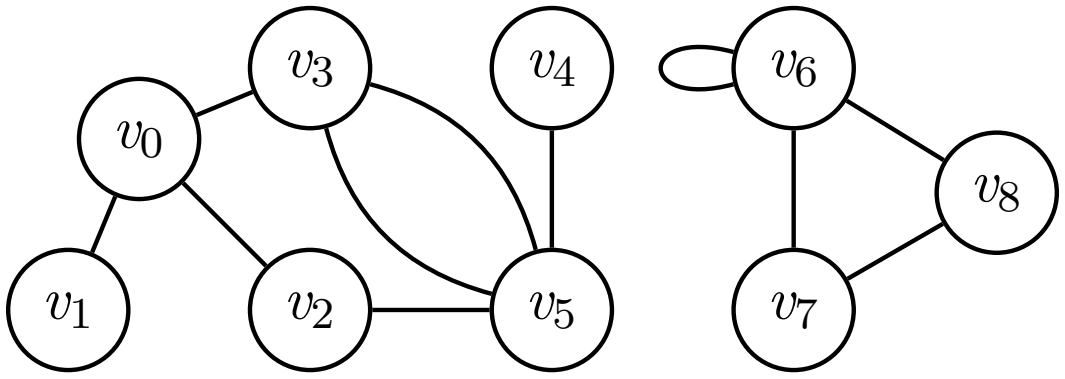
Definition

Two vertices u and v are **neighbors** if and only if there exists an edge $e = (u, v)$ between u and v.

Example

v_0 and v_1 are neighbors, being **adjacent** to one another and **incident** to (v_0, v_1).

Graph Terminology



Definition

A **loop** is an edge $e = (u, v)$ such that $u = v$.

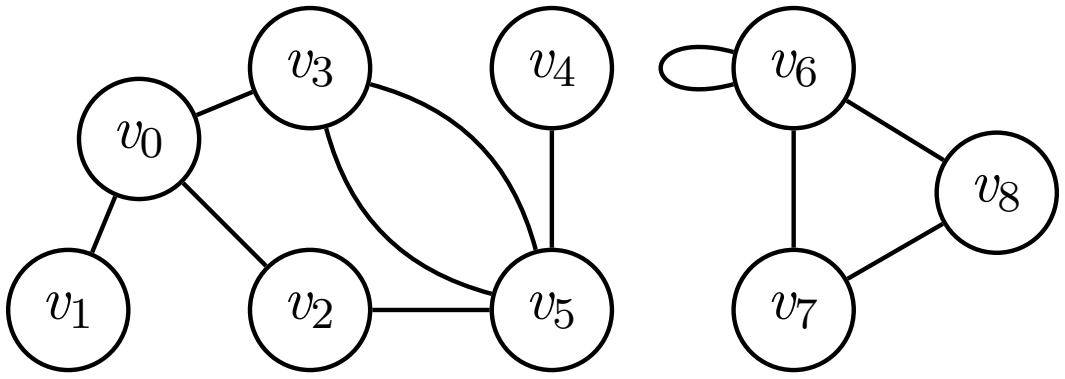
Example

(v_6, v_6) is a loop incident to v_6.

Example

(v_3, v_5) and (v_5, v_3) are not a loop.

Graph Terminology



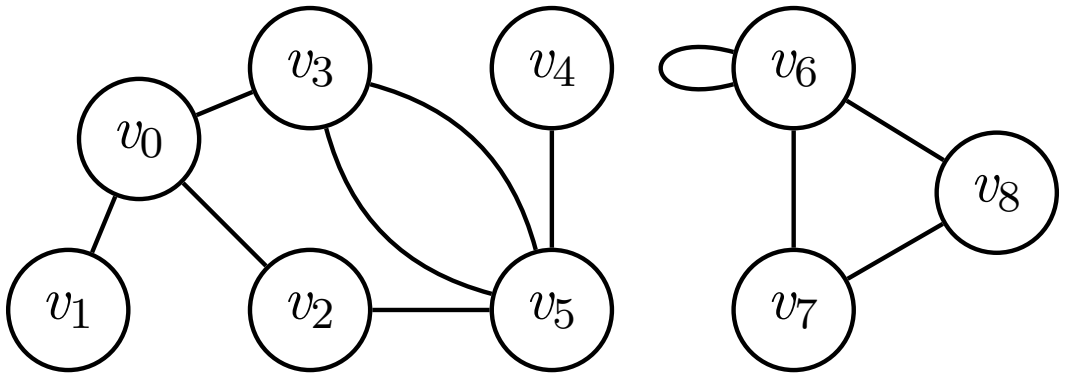
Definition

The **degree** of a vertex v, denoted $\deg(v)$, is the number of edges incident to it, where loops are counted twice.

Example

$\deg(v_1) = 1$, whereas $\deg(v_6) = 4$.

Graph Terminology



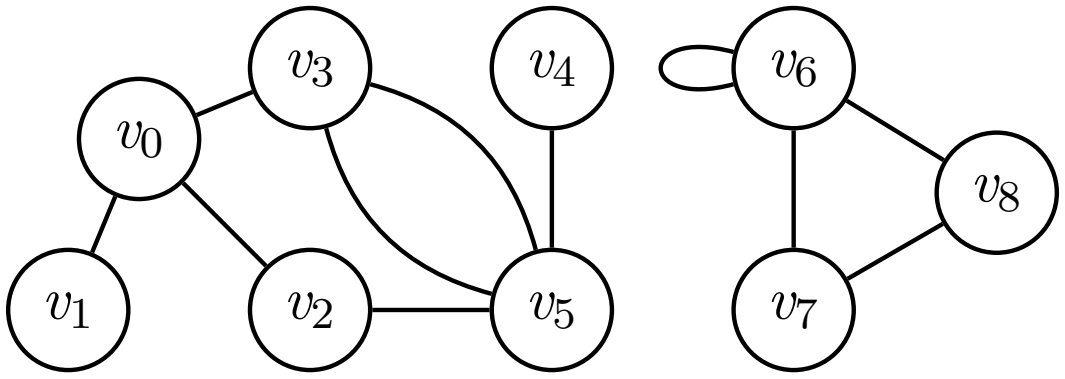
Definition

A **simple graph** contains neither loops nor multiple edges between the same two vertices.

Example

This is not a simple graph: there is a loop incident to v_6, and there are multiple edges between v_3 and v_5.

Graph Terminology



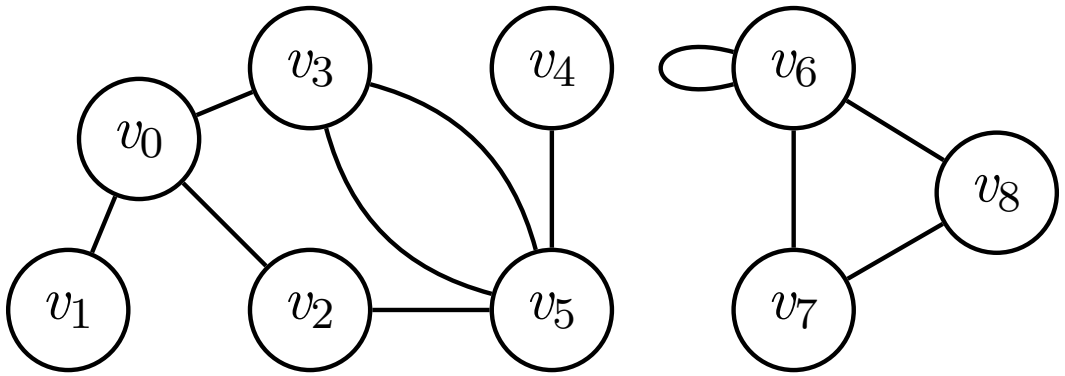
Definition

A **path** is a sequence of vertices $(u, \ldots, v)$ such that, for all consecutive pairs v_i, v_j, there exists an edge between v_i and v_j.

Example

$(v_1, v_0, v_2, v_5, v_3)$ is a path that **traverses** (v_2, v_5) and **passes through** v_0, having **length** 4.

Graph Terminology



Definition

A **cycle** is a path $(u, \ldots, v)$ such that $u = v$, having non-zero length and traversing no edge twice.

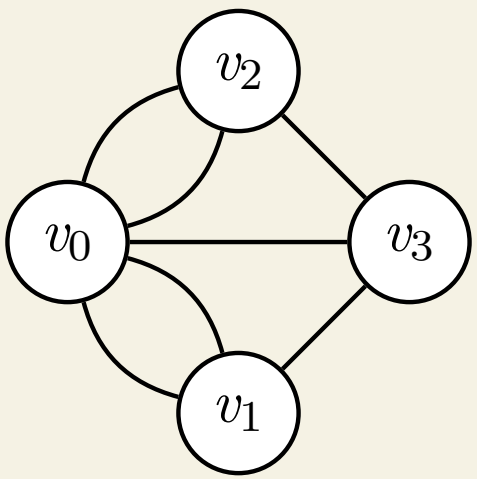
Example

$(v_2, v_5, v_3, v_0, v_2)$ is a cycle, but $(v_2, v_5, v_3, v_5, v_2)$ is not a cycle.

Graph Terminology

Example *(cont.)*

Consider the Seven Bridges of Königsberg:



- Can one walk across every bridge exactly once?
- Does there exist a path that traverses every edge exactly once?

Reachability

Connected Components

Example

- Vertices represent cities.
- Two vertices are connected by an edge $e = (u, v)$ if and only if there is a highway directly between city u and city v.

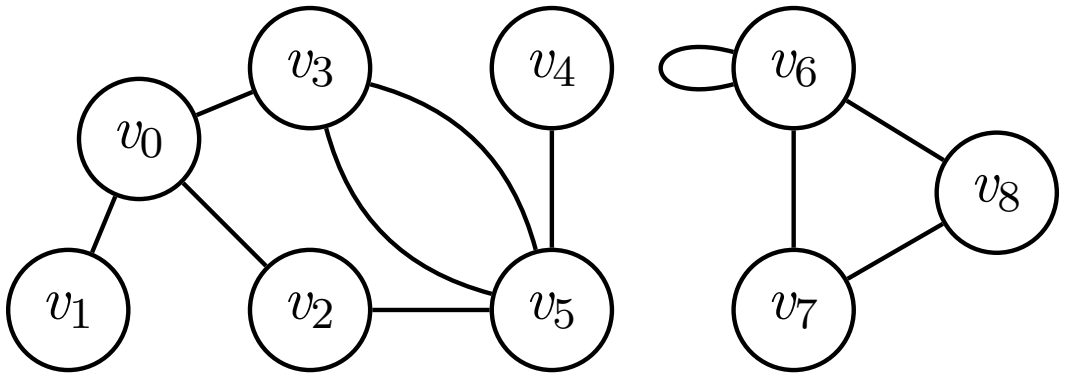
Is it possible to drive from any city to any other city?

Example

- Vertices represent people.
- Two vertices are connected by an edge $e = (u, v)$ if and only if person u and person v make regular face-to-face contact.

If one person contracts measles, is it possible to infect everyone?

Connected Components



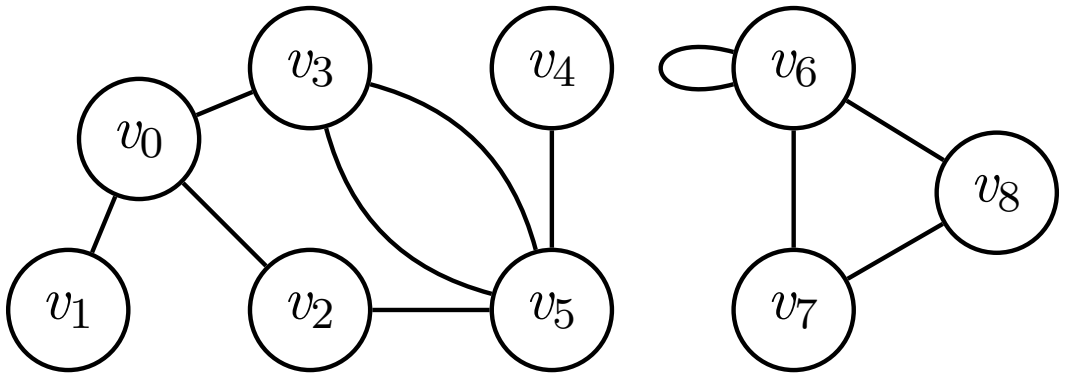
Definition

A vertex v is **reachable** from a vertex u if and only if there exists a path from u to v.

Example

v_3 is reachable from v_1 (and vice versa), but it is not reachable from v_8 (and vice versa).

Connected Components



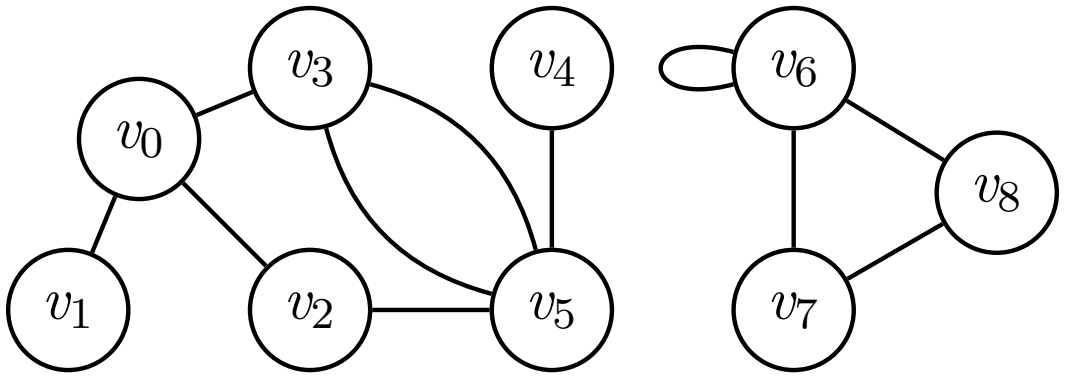
Definition

A graph is **connected** if and only if there exists a path between every pair of distinct vertices.

Example

This graph is not connected: there are (among others) no paths from v_8 to v_3.

Connected Components



Definition

A **connected component** is a maximal connected subgraph.

Example

$\{v_6, v_7, v_8\}$ forms a component, but $\{v_5, v_6, v_7, v_8\}$ does not.

Example

$\{v_0, v_1, v_2, v_3, v_4, v_5\}$ forms a component, but $\{v_0, v_1, v_2\}$ does not.

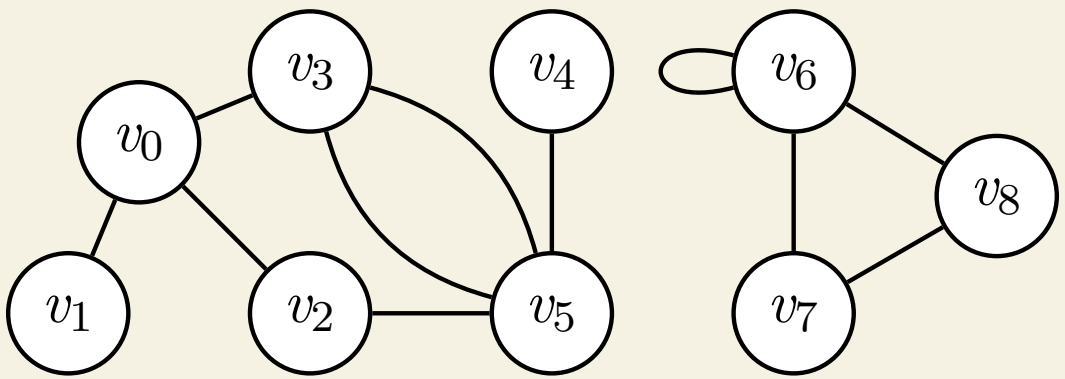
Connected Components

Consider the following problem:

- Given a graph $G = (V, E)$, find the components of G, thereby determining whether or not any v is reachable from any u.

Example

Given:



...the components are $\{v_0, v_1, v_2, v_3, v_4, v_5\}$ and $\{v_6, v_7, v_8\}$.

Depth-First Search

$\textsc{Explore}(G = (V, E), v)$

Input: A graph G and a vertex v, where every vertex is either "explored" or "unexplored"

Output: The previously "unexplored" vertices reachable from v

1: **let** v be "explored" and S be $\{v\}$
2: **for** all neighbors of v, u **do**
3: **if** u is "unexplored" **then**
4: **let** S be $S \cup \textsc{Explore}(G, u)$
5: **return** S

Depth-First Search

DepthFirstSearch$(G = (V, E))$

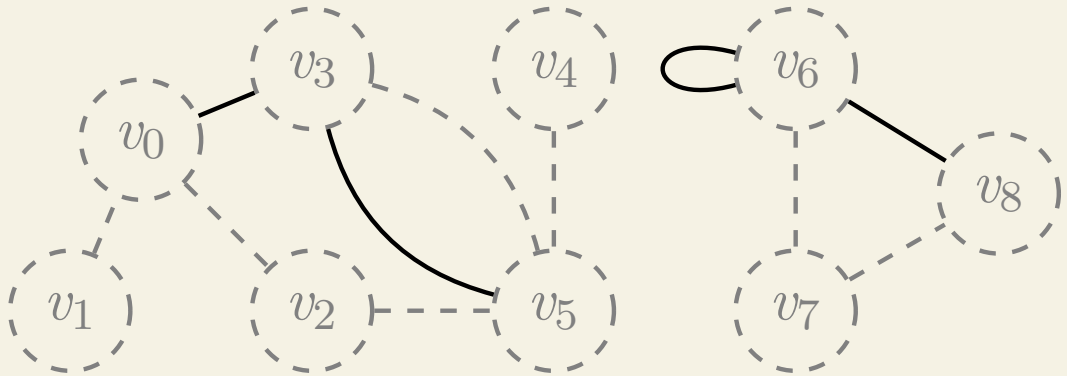
Input: A graph G

Output: The components of G

1: **let** S be $\emptyset$
2: **for** all $v \in V$ **do**
3: **let** v be "unexplored"
4: **for** all $v \in V$ **do**
5: **if** v is "unexplored" **then**
6: **let** S be $S \cup \{\text{Explore}(G, v)\}$
7: **return** S

Depth-First Search

Example



1. Start at v_0.
2. From v_0, explore v_1.
3. From v_0, explore v_2.
4. From v_2, explore v_5.
5. From v_5, explore v_3.
6. From v_5, explore v_4.
7. Start at v_6.
8. From v_6, explore v_7.
9. From v_7, explore v_8.

Depth-First Search

Example

- DepthFirstSearch considers each vertex twice.
- Explore is called once on each vertex.
- Explore collectively considers each edge once or twice.

DepthFirstSearch has complexity $O(|V| + |E|)$.

- A call to Explore(G, v) traces out a **spanning tree** of v's component.
- Calling Explore on every vertex traces out a **spanning forest** of components.

Definition

A **tree** is a connected, undirected, acyclic graph.

Depth-First Search

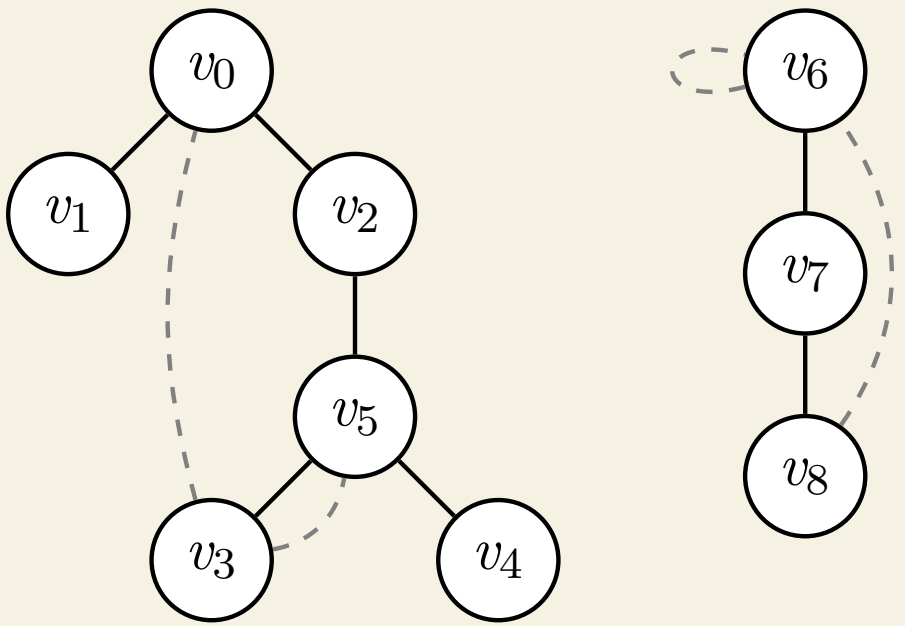
$\textsc{Explore}(G = (V, E), v)$

1: **let** v be "explored", a tree vertex with no children
2: **for** ... **do**
3: **if** ... **then**
4: **let** $\textsc{Explore}(G, u)$ be a subtree of v
5: **return** the tree rooted at v

Depth-First Search

- The edges that appear in a tree are **tree edges**.
- The edges that do not appear in a tree are **back edges**.
- The tree edges and back edges are not necessarily unique.

Example *(cont.)*



Depth-First Search

EXPLORE$(G = (V, E), v)$

1: **let** v be "explored", a tree vertex with no children
2: **let** $\mathrm{pre}(v)$ be the current time; increment the time

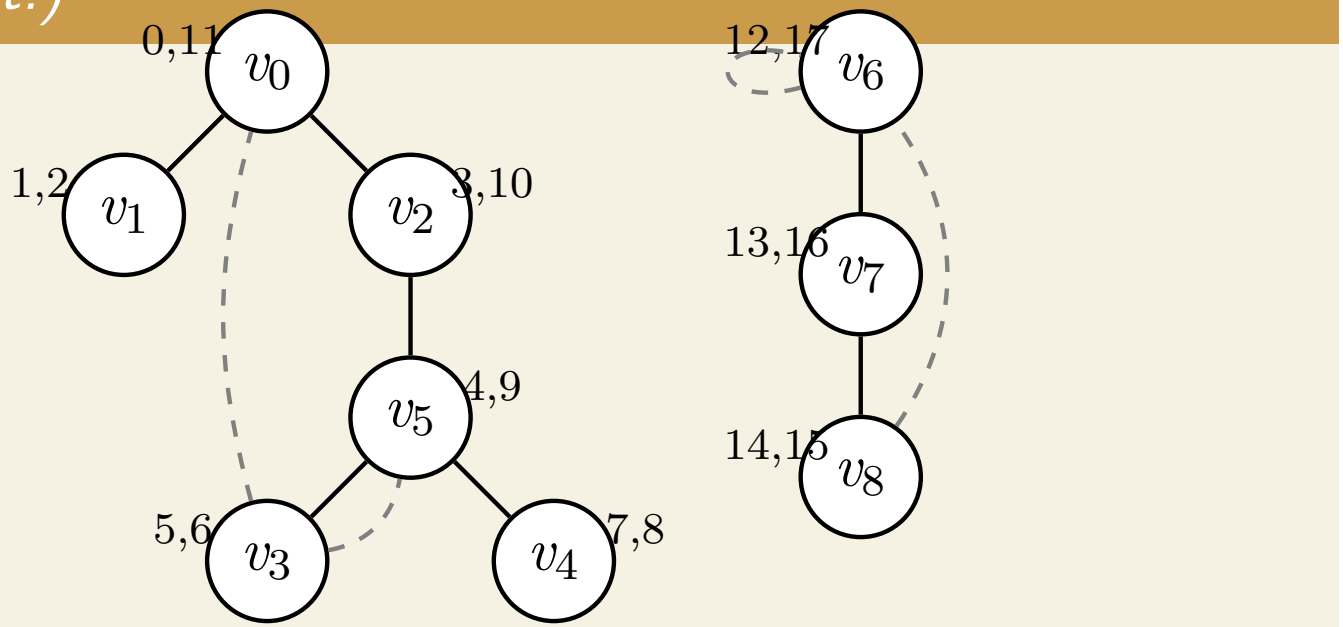
$\vdots$

6: **let** $\mathrm{post}(v)$ be the current time; increment the time
7: **return** the tree rooted at v

Depth-First Search

- A vertex's **previsit number** indicates when exploration begins.
- A vertex's **postvisit number** indicates when exploration ends.

Example *(cont.)*



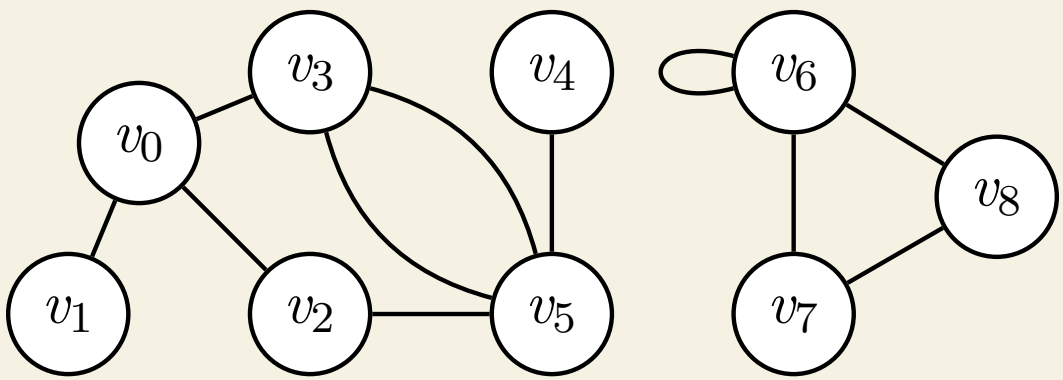
Graph Colorings

Consider the following problem:

- Given a graph $G = (V, E)$, a back edge $e = (u, v)$, and all pre- and postvisit numbers, determine whether or not e creates an odd-length cycle in G.

Example

Given:



...the back edge (v_8, v_6) creates an odd-length cycle.

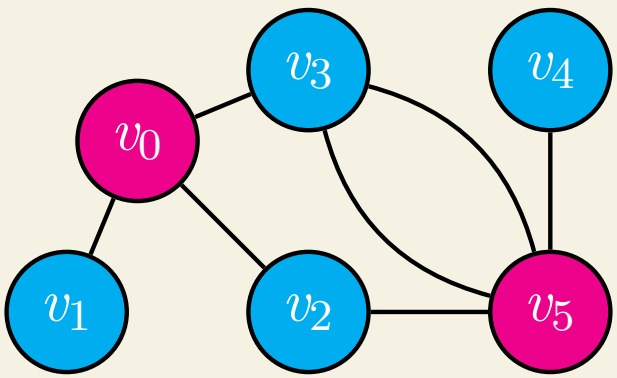
Graph Colorings

Consider the following problem:

- Given a graph, determine whether or not it is **2-colorable**: its vertices can be colored using two colors such that no two adjacent vertices have the same color.

Example

Given:

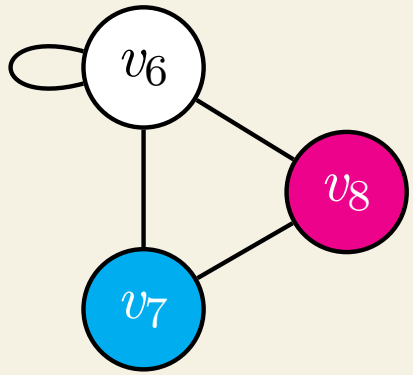


...the graph is 2-colorable.

Graph Colorings

Example *(cont.)*

Given:



...the graph is not 2-colorable.

- The neighbors of a vertex must all have the opposite color.
- The vertices along a path must alternate colors.

Theorem

A graph is 2-colorable if and only if it contains no odd-length cycles.

Directed Paths

Directed Graphs

Example

Some highways are closed in one direction:

- Vertices represent cities.
- Two vertices are connected by an edge $e = (u, v)$ if and only if there is a highway directly from city u to city v.

Example

Some diseases are not equally transmissible between two people:

- Vertices represent people.
- Two vertices are connected by an edge $e = (u, v)$ if and only if person u makes contact that can transmit diseases to person v.

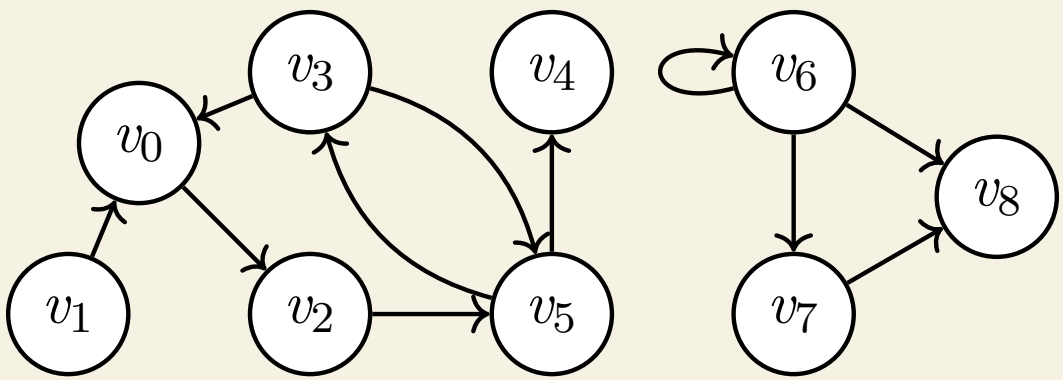
Directed Graphs

Definition

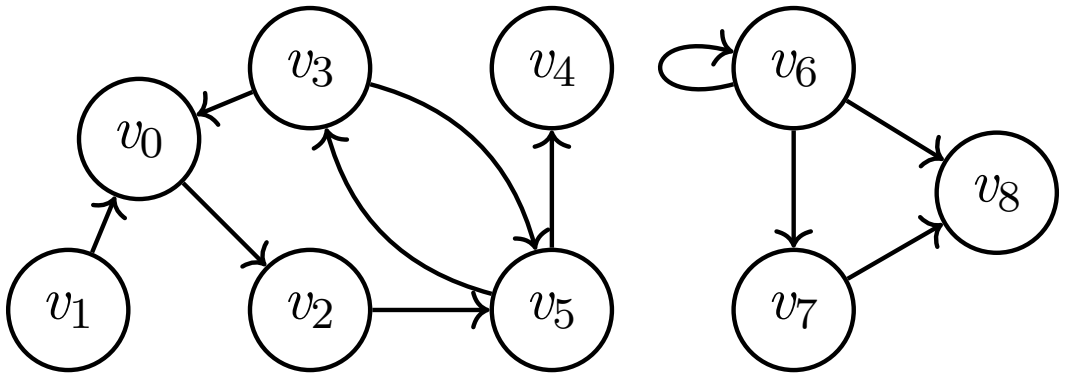
A **directed graph** $G = (V, E)$ is a graph in which each edge $e = (u, v)$ starts at u and ends at v.

- A graph that is not directed is said to be **undirected**.
- Directed graphs are sometimes called "digraphs".

Example



Directed Graph Terminology



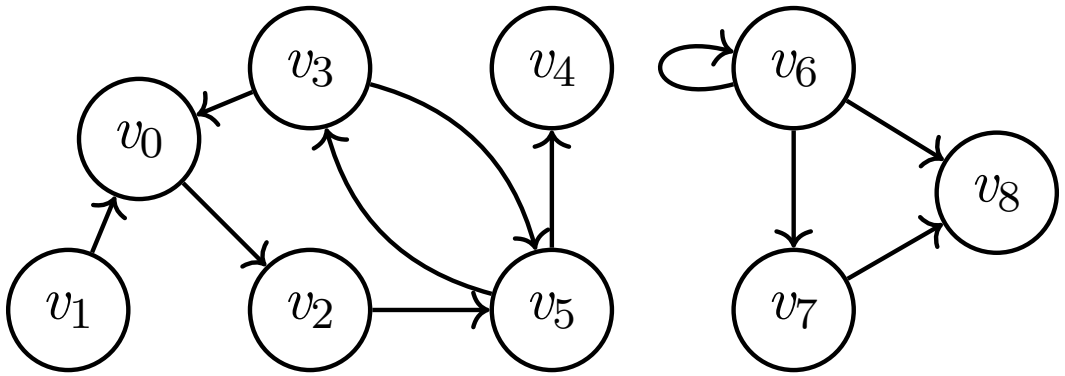
Definition

A vertex u is the **predecessor** of v, and v, the **successor** of u, if and only if there exists an edge $e = (u, v)$ from u to v.

Example

v_1 is a predecessor of v_0, and v_0 is a successor of v_1.

Directed Graph Terminology



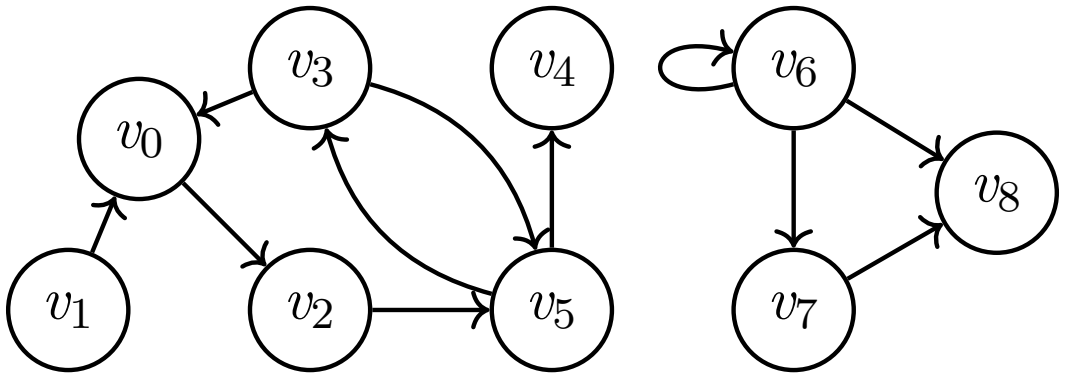
Definition

The **in-degree** of a vertex v, denoted $\deg^-(v)$, is the number of edges directed in to v.

Example

$\deg^-(v_1) = 0$, whereas $\deg^-(v_6) = 1$.

Directed Graph Terminology



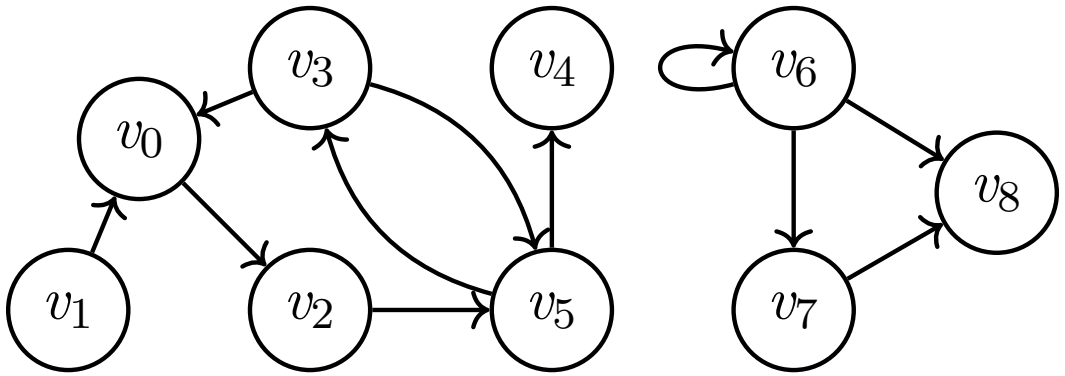
Definition

The **out-degree** of a vertex v, denoted $\deg^+(v)$, is the number of edges directed out of v.

Example

$\deg^+(v_1) = 1$, whereas $\deg^+(v_6) = 3$.

Directed Graph Terminology



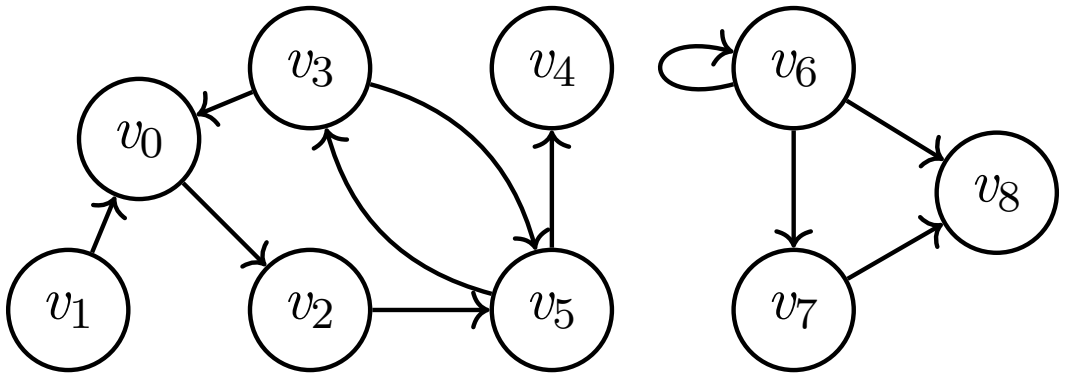
Definition

A **path** is a sequence of vertices $(u, \ldots, v)$ such that, for all consecutive pairs v_i, v_j, there exists an edge from v_i to v_j.

Example

$(v_1, v_0, v_2, v_5, v_3)$ is a path, but $(v_3, v_5, v_2, v_0, v_1)$ is not a path.

Directed Graph Terminology



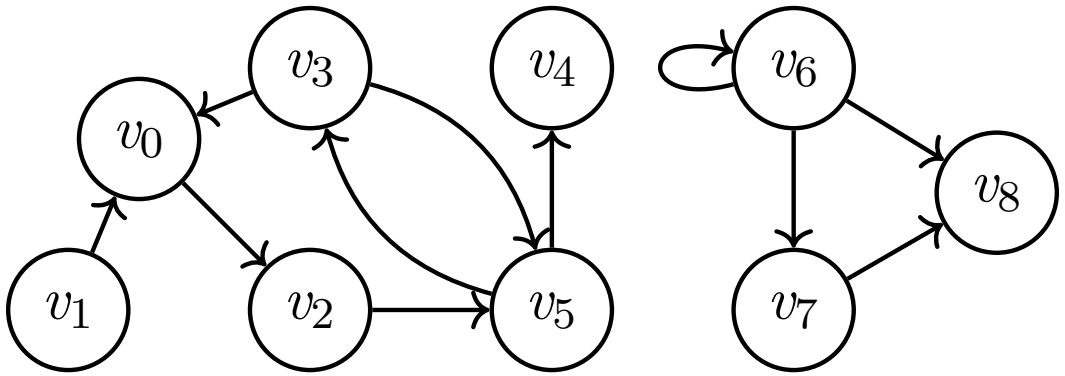
Definition

A directed graph is **strongly connected** if and only if there exist paths between every pair of distinct vertices.

Definition

A directed graph is **weakly connected** if and only if it is not strongly connected, but its underlying undirected graph is connected.

Directed Graph Terminology



Definition

A **strongly connected component** is a maximal strongly connected subgraph.

Example

$\{v_0, v_2, v_3, v_5\}$ forms a strongly connected component, but $\{v_0, v_1, v_2, v_3, v_4, v_5\}$ does not.

Directed Depth-First Search

DirectedExplore$(G = (V, E), v)$

Input: A directed graph G, and a vertex v, where every vertex is either "explored" or "unexplored"

Output: A tree of previously "unexplored" vertices reachable from v

1: **let** v be "explored", a tree vertex with no children
2: **let** $\mathrm{pre}(v)$ be the current time; increment the time
3: **for** all successors of v, u **do**
4: **if** u is "unexplored" **then**
5: **let** DirectedExplore(G, u) be a subtree of v
6: **let** $\mathrm{post}(v)$ be the current time; increment the time
7: **return** the tree rooted at v

Directed Depth-First Search

DirectedDepthFirstSearch$(G = (V, E))$

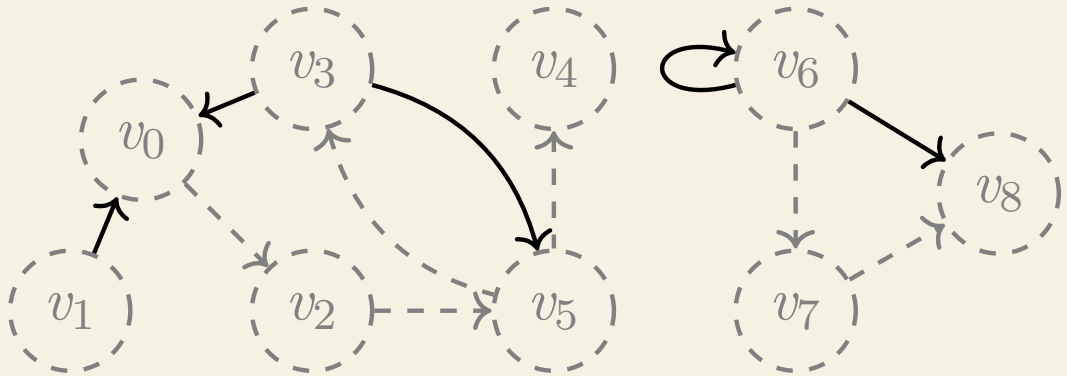
Input: A directed graph G

Output: A forest of vertices in G

1: **let** S be $\emptyset$
2: **for** all $v \in V$ **do**
3: **let** v be "unexplored"
4: **for** all $v \in V$ **do**
5: **if** v is "unexplored" **then**
6: **let** S be $S \cup \{\text{DirectedExplore}(G, v)\}$
7: **return** S

Directed Depth-First Search

Example

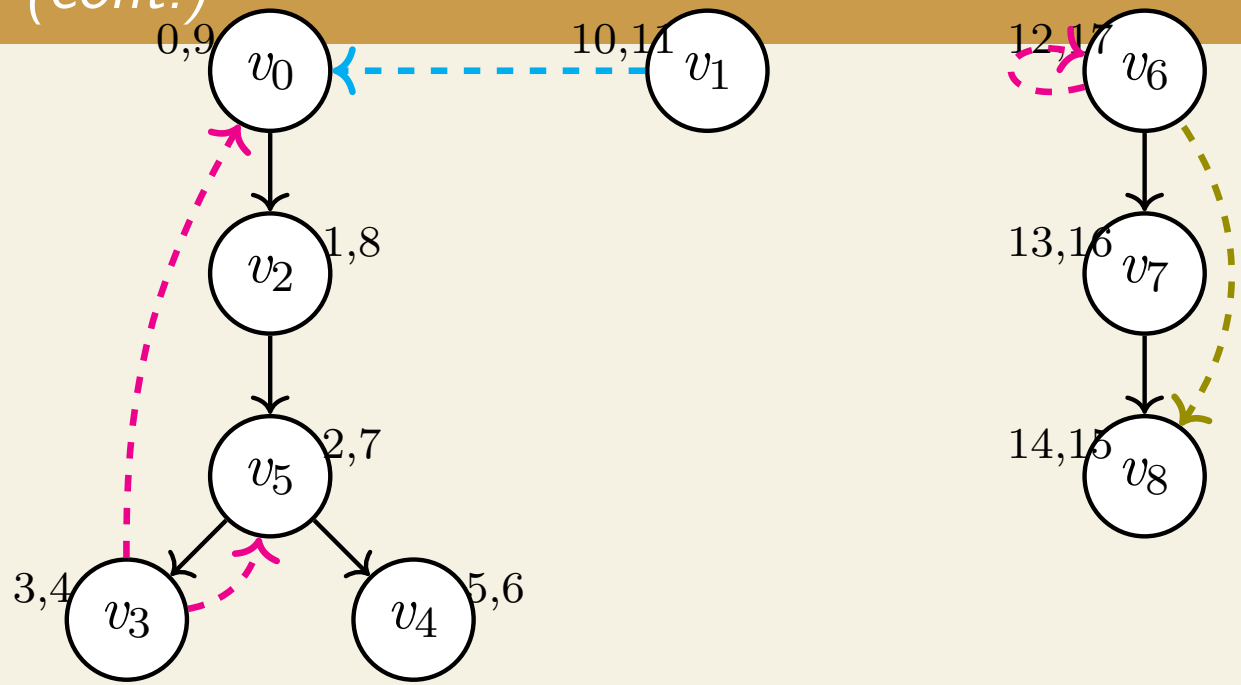


1. Start at v_0.
2. From v_0, explore v_2.
3. From v_2, explore v_5.
4. From v_5, explore v_3.
5. From v_5, explore v_4.
6. Start at v_1.
7. Start at v_6.
8. From v_6, explore v_7.
9. From v_7, explore v_8.

Directed Depth-First Search

- **Tree edges**
- **Forward edges**
- **Back edges**
- **Cross edges**

Example *(cont.)*



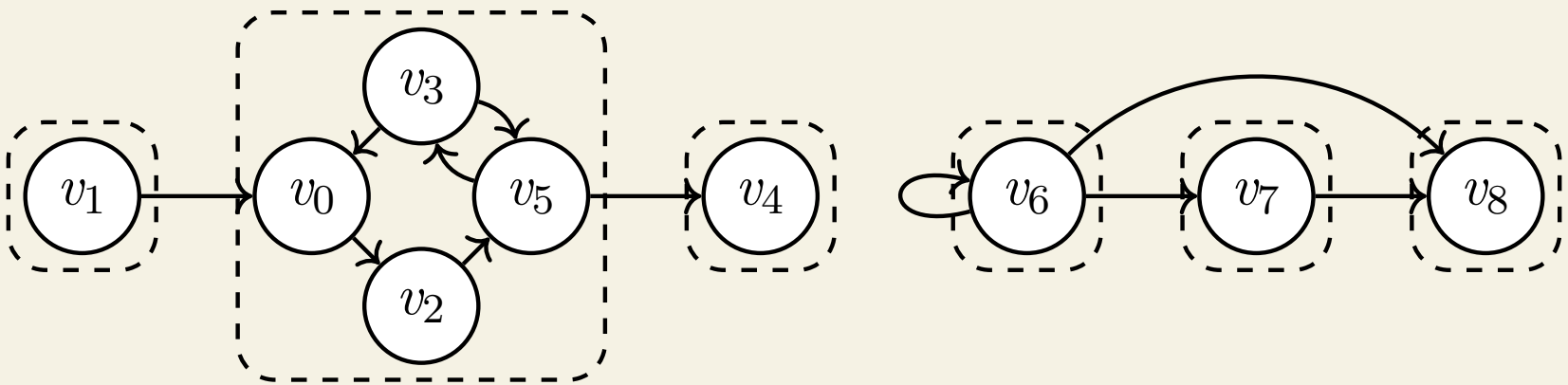
Kosaraju's Algorithm

Definition

A **directed acyclic graph**, or "DAG", is a directed graph containing no cycles.

- Every directed graph is a DAG of its strongly connected components.

Example



Kosaraju's Algorithm

- If u is in a **sink** component and $\textsc{DirectedExplore}(G, u)$ reaches v, then u and v are in the same component.

Definition

The **transpose** of a directed graph $G = (V, E)$, denoted G^T, is a directed graph where $(u, v) \in E^T$ if and only if $(v, u) \in E$.

- The components of G are the components of G^T.
- The sink components of G are the **source** components of G^T.

Theorem

If v is a vertex such that $\text{post}(v)$ is maximized, then v is in a source component.

Kosaraju's Algorithm

$\textsc{TransposedExplore}(G = (V, E), v)$

Input: A directed graph G and a vertex v, where every vertex is either "explored" or "unexplored"

Output: The previously "explored" vertices reachable from v in G^T, sorted in descending order by postvisit number

1: **let** v be "unexplored" and S be $\emptyset$
2: **for** all predecessors of v, u **do**
3: **if** u is "explored" **then**
4: **let** S be $\textsc{TransposedExplore}(G, u) +\!\!+ S$
5: **return** $(v) +\!\!+ S$

Kosaraju's Algorithm

$\textsc{StrongComponents}(G = (V, E))$

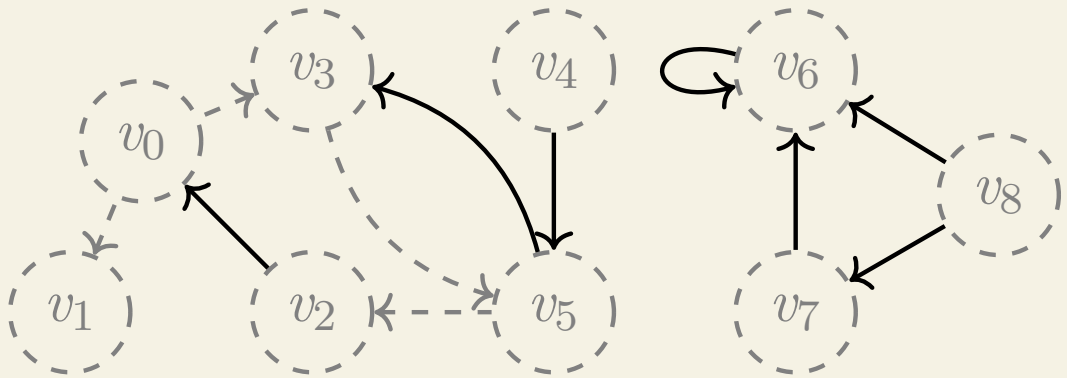
Input: A directed graph G

Output: The strongly connected components of G

```
1:  let S be ∅ and T be ∅
2:  for all v ∈ V do
3:      let v be "explored"
4:  for all v ∈ V do
5:      if v is "explored" then
6:          let S be TransposedExplore(G, v) +# S
7:  for all v ∈ S do
8:      if v is "unexplored" then
9:          let T be T ∪ {DirectedExplore(G, v)}
10: return T
```

Kosaraju's Algorithm

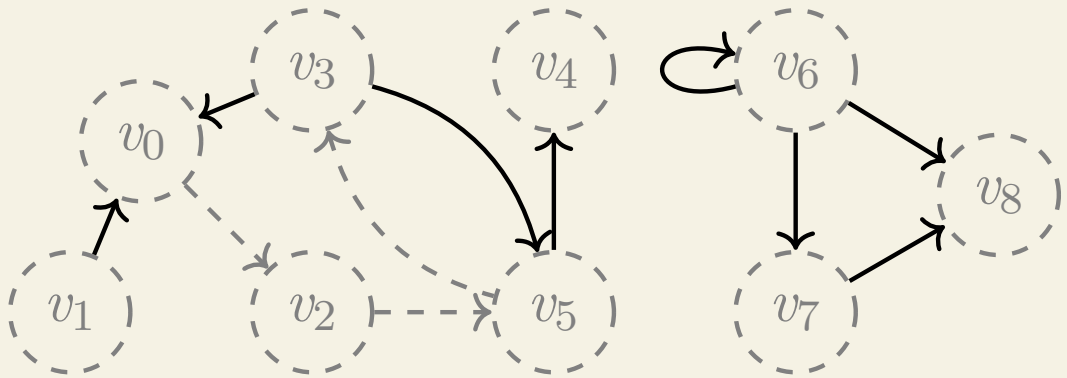
Example *(cont.)*



1. Start at v_0.
2. From v_0, explore v_1.
3. From v_0, explore v_3.
4. From v_3, explore v_5.
5. From v_5, explore v_2.
6. Start at v_4.
7. Start at v_6.
8. Start at v_7.
9. Start at v_8.

Kosaraju's Algorithm

Example *(cont.)*



10. Start at v_8.
11. Start at v_7.
12. Start at v_6.
13. Start at v_4.
14. Start at v_0.
15. From v_0, explore v_2.
16. From v_2, explore v_5.
17. From v_5, explore v_3.
18. Start at v_1.

Kosaraju's Algorithm

Example

- StrongComponents considers each vertex thrice.
- TransposedExplore is called once on each vertex.
- TransposedExplore collectively considers each edge once.
- DirectedExplore is called once on each vertex.
- DirectedExplore collectively considers each edge once.

StrongComponents has complexity $O(|V| + |E|)$.

Shortest Paths

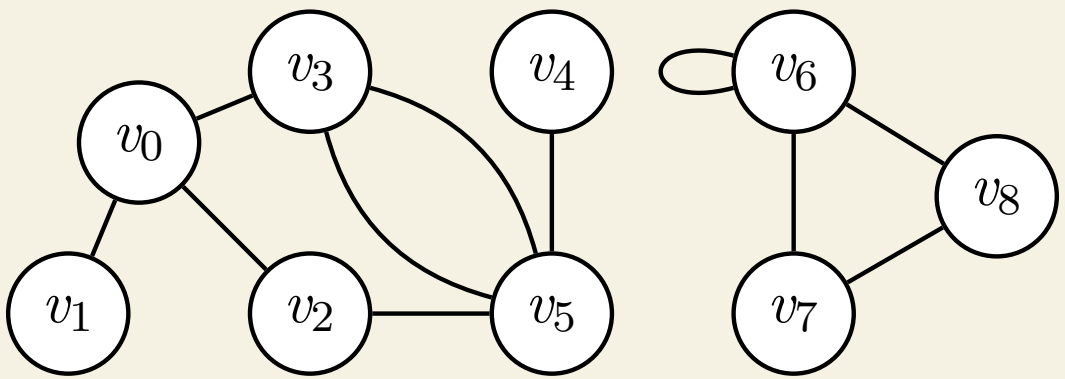
Breadth-First Search

Consider the following problem:

- Given a graph $G = (V, E)$ and a vertex s, find the shortest paths from s to all other vertices.

Example

Given:



...the shortest path from v_1 to v_3 is (v_1, v_0, v_3).

Breadth-First Search

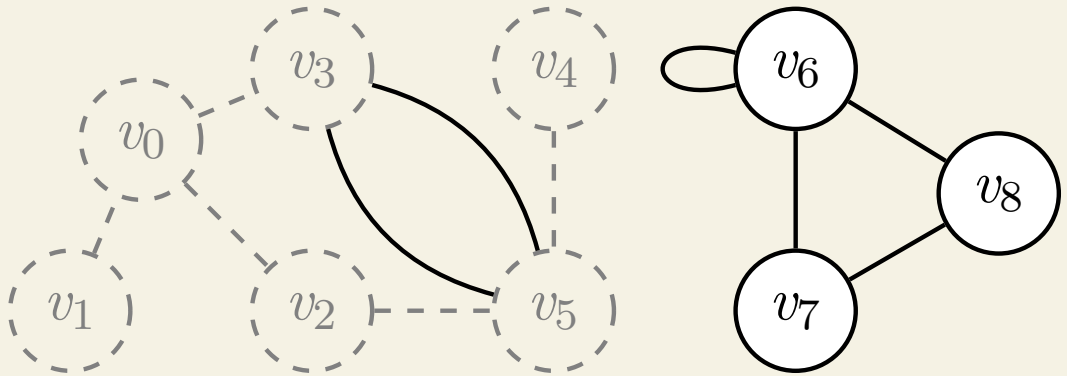
BreadthFirstSearch$(G = (V, E), s)$

Input: A graph G and a starting vertex s
Output: The distances from s to all vertices

```
1:  for all v ∈ V do
2:      let dist(v) be ∞
3:  let dist(s) be 0 and Q be a queue containing s
4:  while Q is not empty do
5:      Dequeue a vertex from Q, v
6:      for all neighbors of v, u do
7:          if dist(u) = ∞ then
8:              let dist(u) be dist(v) + 1
9:              Enqueue u to Q
10: return dist(v) for all v ∈ V
```

Breadth-First Search

Example



1. Start at v_0, $\text{dist}(v_0) = 0$.
2. From v_0, $\text{dist}(v_1) = 1$.
3. From v_0, $\text{dist}(v_2) = 1$.
4. From v_0, $\text{dist}(v_3) = 1$.
5. From v_2, $\text{dist}(v_5) = 2$.
6. From v_5, $\text{dist}(v_4) = 3$.
7. Note $\text{dist}(v_6) = \infty$.
8. Note $\text{dist}(v_7) = \infty$.
9. Note $\text{dist}(v_8) = \infty$.

Breadth-First Search

Example

- BreadthFirstSearch initially considers each vertex once.
- BreadthFirstSearch explores each vertex at most once.
- BreadthFirstSearch considers each edge once or twice.

BreadthFirstSearch has complexity $O(|V| + |E|)$.

- Like a depth-first search, a breadth-first search never explores a vertex twice or traverses an edge thrice.
- Unlike a depth-first search, a breadth-first search stores vertices in a queue, rather than on the runtime stack.

Weighted Graphs

Example

Some highways are longer than others:

- Vertices represent cities.
- Two vertices are connected by an edge $e = (u, v)$ of weight w_e iff there is a highway of length w_e between city u and city v.

Example

Some diseases are more contagious with prolonged contact:

- Vertices represent people.
- Two vertices are connected by an edge $e = (u, v)$ of weight w_e iff person u and person v make contact for w_e time.

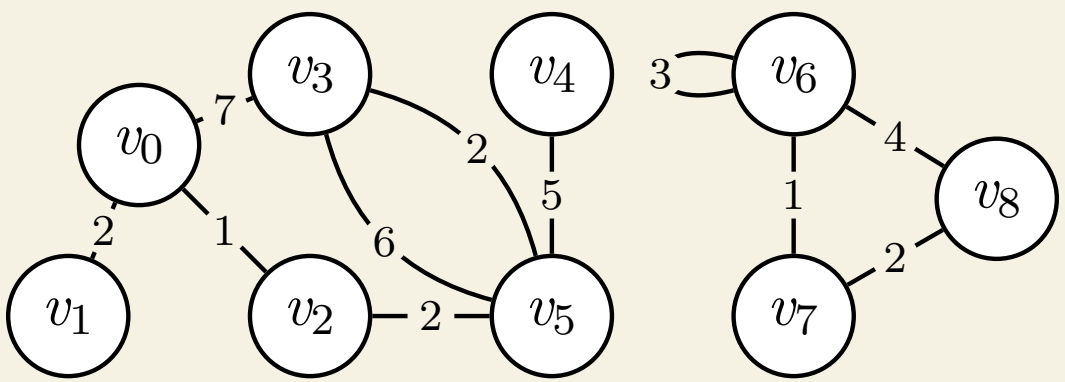
Weighted Graphs

Definition

A **weighted graph** $G = (V, E)$ is a graph in which each edge $e = (u, v)$ has a numerical weight w_e.

- A graph that is not weighted is said to be **unweighted**.
- Edge weights are often restricted to strictly positive integers.

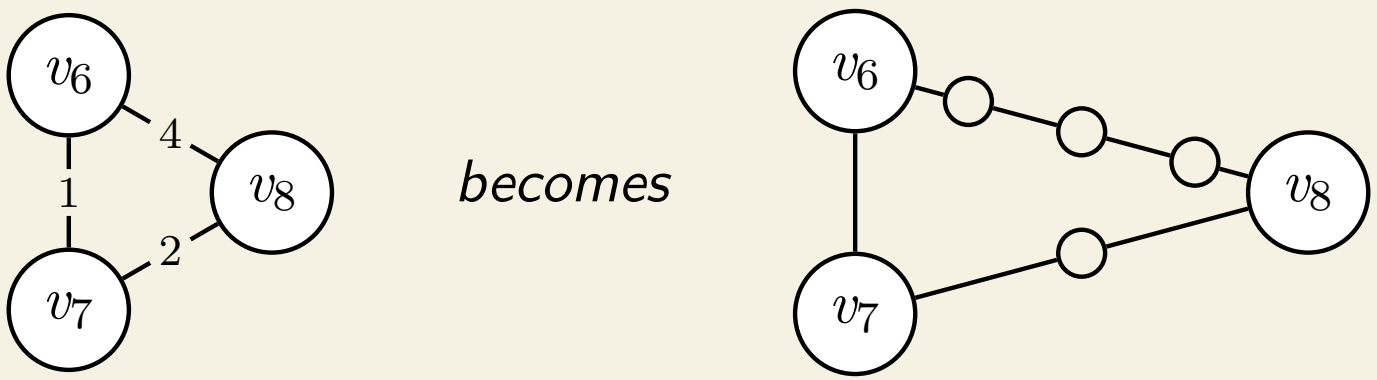
Example



Dijkstra's Algorithm

- A breadth-first search prioritizes vertices by number of edges, rather than by sum of edges' weights.
- If its weights are strictly positive integers, then a weighted graph can be made unweighted by adding "dummy vertices".

Example



Dijkstra's Algorithm

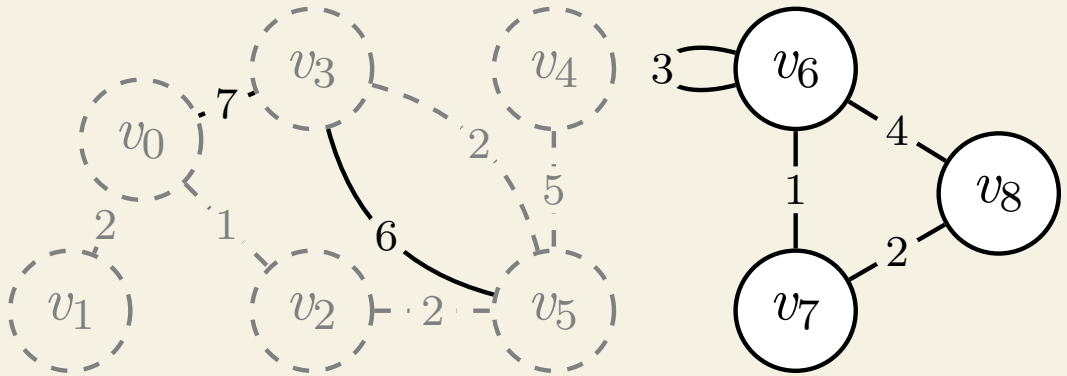
$\textsc{ShortestPath}(G = (V, E), s)$

Input: A weighted graph G and a starting vertex s
Output: The distances from s to all vertices

```
1:  for all v ∈ V do
2:      let dist(v) be ∞
3:  let dist(s) be 0, Q be a min-priority queue containing all v ∈ V
4:  while Q is not empty do
5:      Dequeue a vertex from Q, v
6:      for all edges e = (v, u) do
7:          if dist(u) > dist(v) + w_e then
8:              let dist(u) be dist(v) + w_e
9:              Update u in Q
10: return dist(v) for all v ∈ V
```

Dijkstra's Algorithm

Example



1. Start at v_0, $\text{dist}(v_0) = 0$.
2. Consider v_2, $\text{dist}(v_2) = 1$.
3. Consider v_1, $\text{dist}(v_1) = 2$.
4. Consider v_5, $\text{dist}(v_5) = 3$.
5. Consider v_3, $\text{dist}(v_3) = 5$.
6. Consider v_4, $\text{dist}(v_4) = 8$.
7. Note $\text{dist}(v_6) = \infty$.
8. Note $\text{dist}(v_7) = \infty$.
9. Note $\text{dist}(v_8) = \infty$.

Dijkstra's Algorithm

Example

- ShortestPath initially considers each vertex once.
- ShortestPath enqueues and dequeues each vertex once.
- ShortestPath updates vertices at most $|E|$ times.

Assuming a binary heap implementation of the min-priority queue, ShortestPath has complexity $O(|V| + |E| \log |V|)$.

- With an array-based implementation of the queue, Dijkstra's algorithm has complexity $O\left(|V|^2\right)$.
- With a Fibonacci heap implementation of the queue, Dijkstra's algorithm has complexity $O(|V| \log |V| + |E|)$.

Eulerian and Hamiltonian Paths

Eulerian Paths

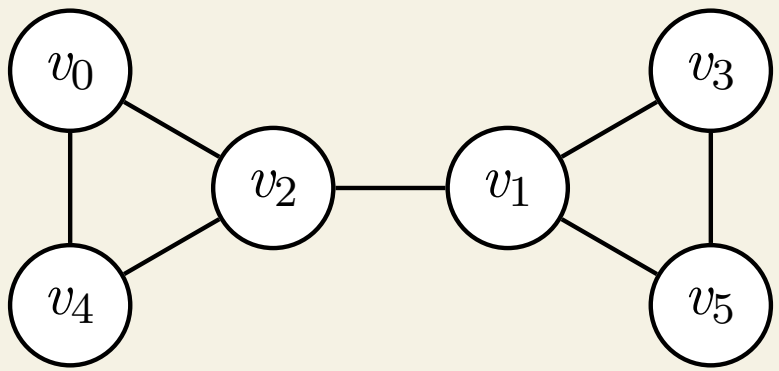
Definition

An **Eulerian path** traverses every edge exactly once.

- An **Eulerian cycle** is an Eulerian path that is a cycle.

Example

Given:



...an Eulerian path is $(v_1, v_3, v_5, v_1, v_2, v_0, v_4, v_2)$. There are no Eulerian cycles.

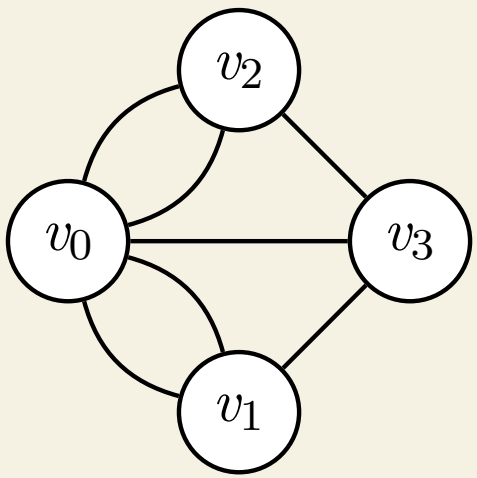
Eulerian Paths

Theorem

A connected graph contains an Eulerian path if and only if it contains 0 or 2 vertices of odd degree.

- If there are exactly 0 vertices of odd degree, the graph contains an Eulerian cycle, which is also a path.

Example



- The Seven Bridges of Königsberg contain 4 vertices of odd degree.
- Thus, it is impossible to walk across every bridge exactly once.

Hierholzer's Algorithm

$\textsc{EulerianPath}(G = (V, E), s, t)$

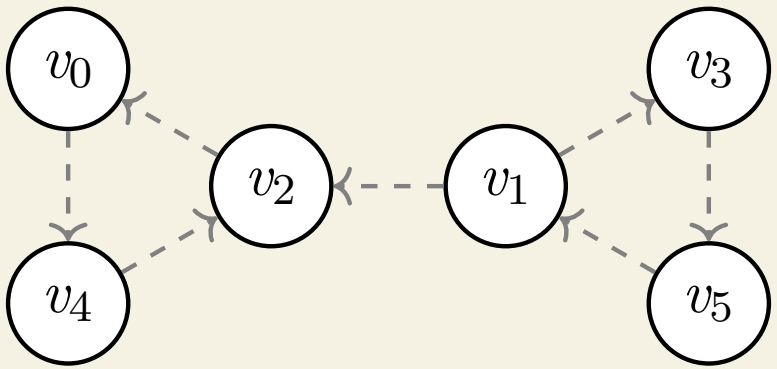
Input: A graph G with two vertices of odd degree s and t, or no vertices of odd degree where $s = t$

Output: A maximal path from s to t traversing no edge twice, with all traversed edges having been removed from G

1: **if** $\deg(s) = 0$ **then**
2: **return** (s)
3: **else**
4: Remove any edge (s, v) from G
5: **let** P be $\textsc{EulerianPath}(G, v, t)$
6: **if** $\deg(s) = 0$ **then**
7: **return** $(s) + P$
8: **else**
9: **return** $\textsc{EulerianPath}(G, s, s) + P$

Hierholzer's Algorithm

Example



1. Start at v_1.
2. Traverse (v_1, v_2) seeking v_2.
3. Traverse (v_2, v_0) seeking v_2.
4. Traverse (v_0, v_4) seeking v_2.
5. Traverse (v_4, v_2) seeking v_2.
6. Return to v_1.
7. Traverse (v_1, v_3) seeking v_1.
8. Traverse (v_3, v_5) seeking v_1.
9. Traverse (v_5, v_1) seeking v_1.

Hierholzer's Algorithm

Example

- EulerianPath performs no initial setup.
- EulerianPath is called on each vertex at most $\frac{\deg(v)}{2}$ times.
- EulerianPath collectively considers each edge exactly once.

EulerianPath has complexity $O(|E|)$.

- If there exists an Eulerian path in a connected graph, then $|E| \geq |V| - 1$.
- Initially counting the vertices of odd degree generally has complexity $O(|V| + |E|)$.

Hamiltonian Paths

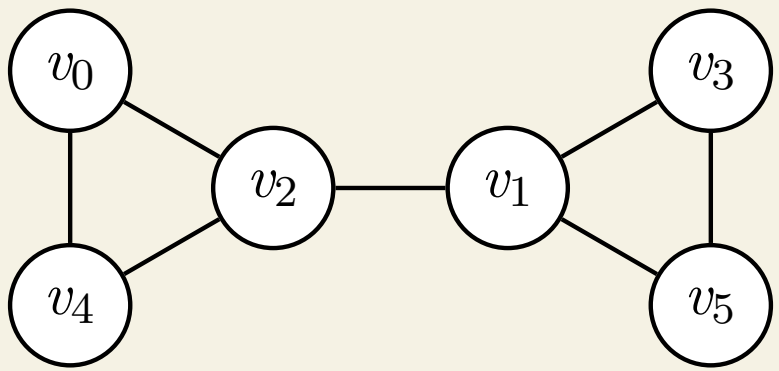
Definition

A **Hamiltonian path** passes through every vertex exactly once.

- A **Hamiltonian cycle** is a Hamiltonian path that is a cycle.

Example

Given:



...a Hamiltonian path is $(v_0, v_4, v_2, v_1, v_3, v_5)$. There are no Hamiltonian cycles.

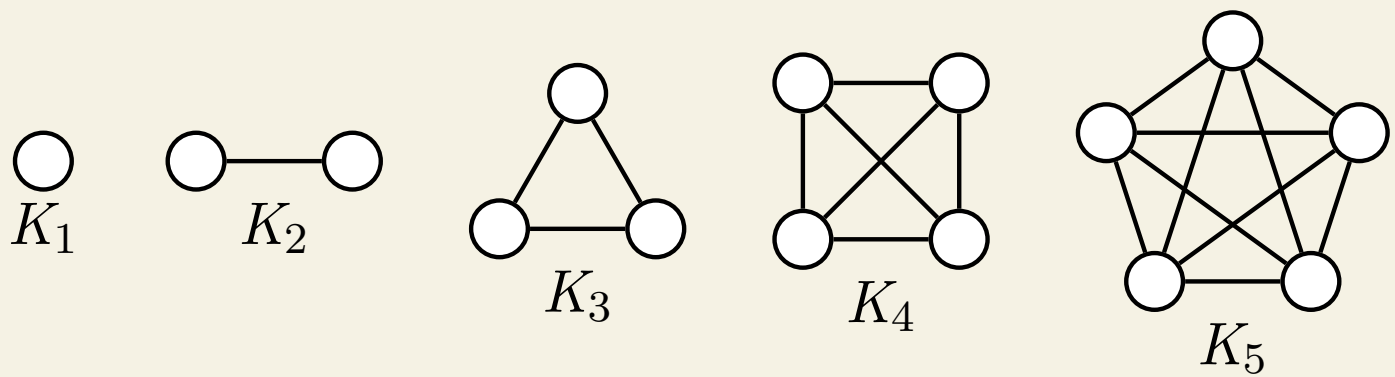
Hamiltonian Paths

Definition

A **complete graph** on n vertices, denoted K_n, is a simple graph containing exactly one edge between every distinct pair of vertices.

- In a complete graph, any permutation of V is a valid path.
- For $n \geq 3$, all complete graphs contain Hamiltonian cycles.

Example



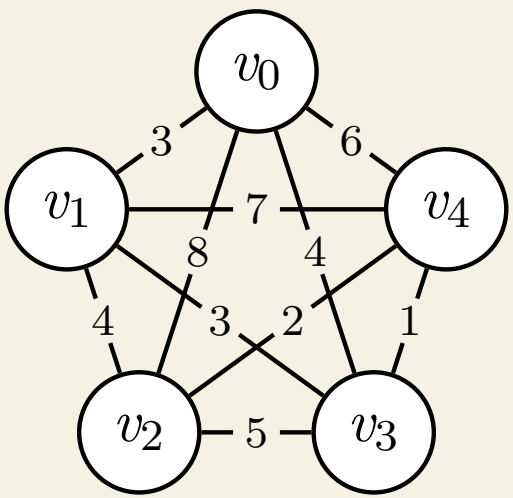
The Traveling Salesperson Problem

Consider the following problem:

- Given a complete, weighted graph, find the Hamiltonian cycle of minimum weight.

Example

Given:



...the Hamiltonian cycle of minimum weight is $(v_0, v_1, v_2, v_4, v_3, v_0)$.

The Traveling Salesperson Problem

NaïveTravelingSalesperson($G = (V, E)$)

Input: A complete, weighted graph G

Output: The minimum weight of a Hamiltonian cycle in G

1: **let** W_A be ∞
2: **for** all permutations of V, $P = (p_0, p_1, \ldots, p_{n-1})$ **do**
3: **let** $e = (p_{n-1}, p_0)$ be an edge and W_B be w_e
4: **for** i **from** 0 **to** $n - 2$ **do**
5: **let** $e = (p_i, p_{i+1})$ be an edge and W_B be $W_B + w_e$
6: **let** W_A be $\min\{ W_A, W_B\}$
7: **return** W_A

The Traveling Salesperson Problem

Example

- There are $|V|!$ permutations of V.
- Every permutation of V is a path in a complete graph G.
- Each permutation of V requires traversing $|V|$ edges.

NaïveTravelingSalesperson has complexity $O(|V|!)$.

- There is no known general mathematical characterization of Hamiltonian paths or cycles.
- As of 1962, the best known traveling salesperson algorithm has complexity $O\left(|V|^2 \cdot 2^{|V|}\right)$.