CSE221 Data Structures
Lecture 19: Graphs

Antoine Vigneron
antoine@unist.ac.kr

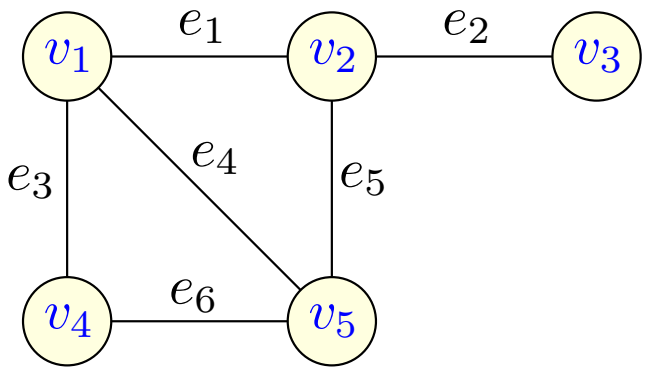
Ulsan National Institute of Science and Technology

November 22, 2021

Introduction

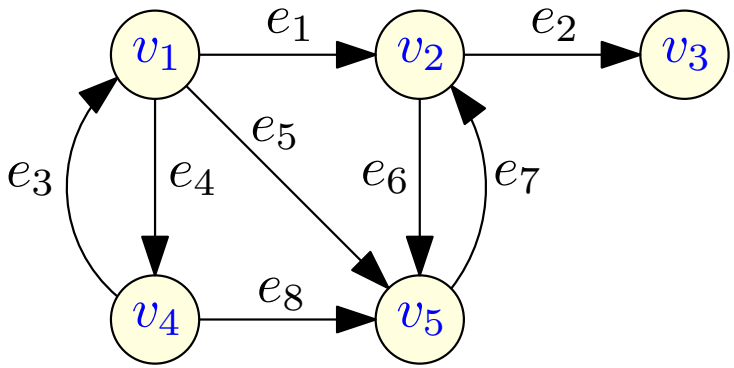
- Final exam is on Wednesday 15 December, 20:00–22:00.
- Assignment 3 is due on Thursday.
- I just modified "test2.cpp", to fix a runtime error that could occur with some compilers.
- Reference for this lecture: Textbook Chapter 13.1, 13.2.

Undirected Graphs



- A *graph* $G = (V, E)$ with *vertex set* $V = \{v_1, v_2, v_3, v_4, v_5\}$ and *edge set* $E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$, where $e_1 = (v_1, v_2)$, $e_2 = (v_2, v_3)$, $e_3 = (v_1, v_4)$, $e_4 = (v_1, v_5)$, $e_5 = (v_2, v_6)$ and $e_6 = (v_4, v_5)$.
- The graph above is *undirected* so the pairs $(u, v) \in E$ are unordered.
- These pairs are sometimes denoted $\{u, v\}$ but we will rather write (u, v). So for undirected graphs, (u, v) and (v, u) refer to the same edge.

Directed Graphs



- A *directed graph* $G = (V, E)$: the pairs $(u, v) \in E$ are *ordered*.
- So we have $e_3 = (v_4, v_1)$, $e_4 = (v_1, v_4)$ and $e_3 \neq e_4$.
- We also have $e_2 = (v_2, v_3) \in E$, but $(v_3, v_2) \notin E$.

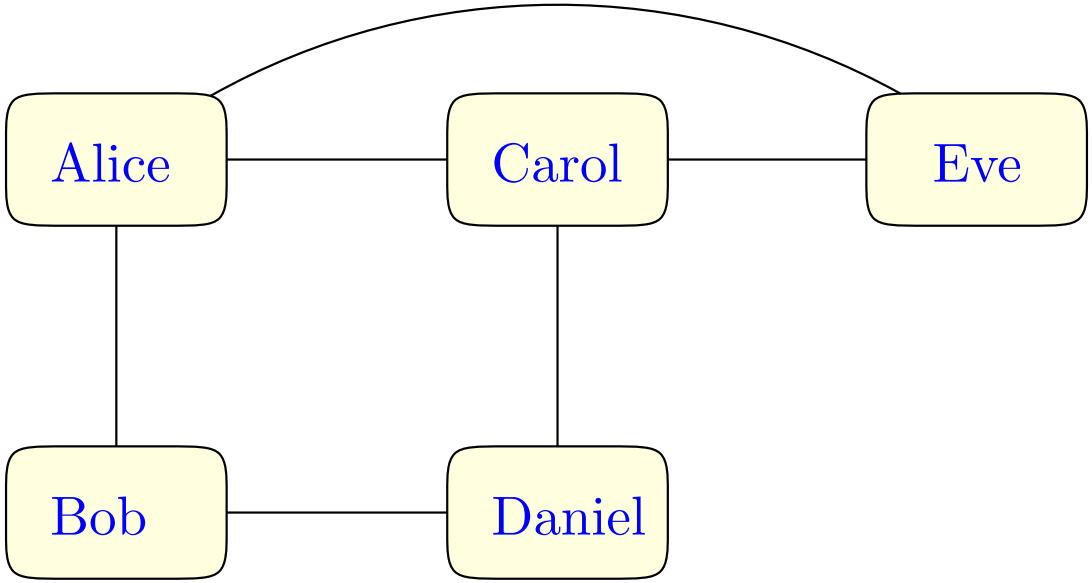
Definition

Definition

A *graph* G consists of a set of *vertices* V and a set of *edges* E. The edges are pairs of vertices. The graph is said to be *directed* if these pairs are ordered, and undirected otherwise.

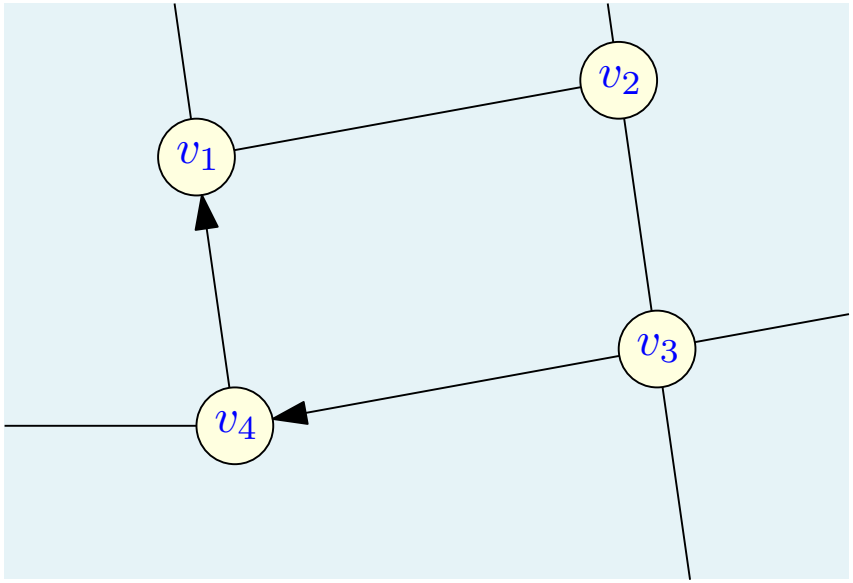
- Directed graphs are also called *digraphs*.
- There are also *mixed graphs* having both directed and undirected edges.
- Often, the number of vertices is denoted n and the number of edges is m. So we have $|V| = n$ and $|E| = m$.

Example



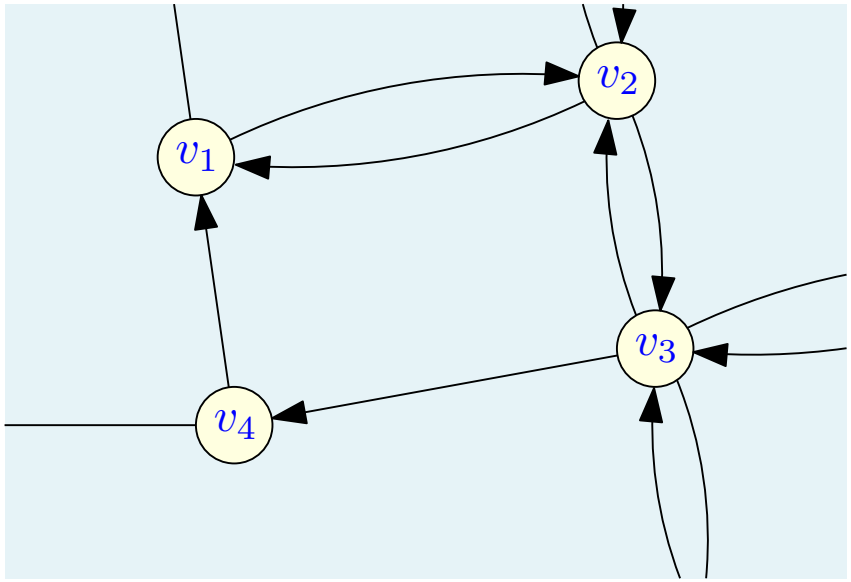
- A social network, where friends are connected through an edge.
- For instance, Alice and Carol are friends, but Carol and Bob are not.

Example



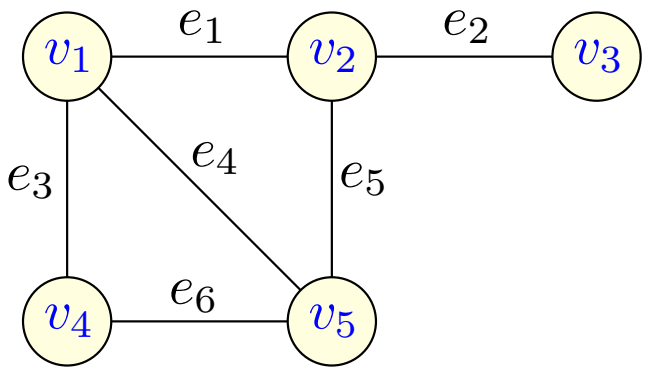
- A road network can be represented by a mixed graph: one-way streets are directed, and two-way streets are undirected.

Example



- It can also be represented by a digraph, if we replace each undirected edge with two directed edges of opposite direction.
- In fact a mixed graph can always be represented by a digraph in this way, so we won't need an extra data structure for mixed graphs.

Terminology

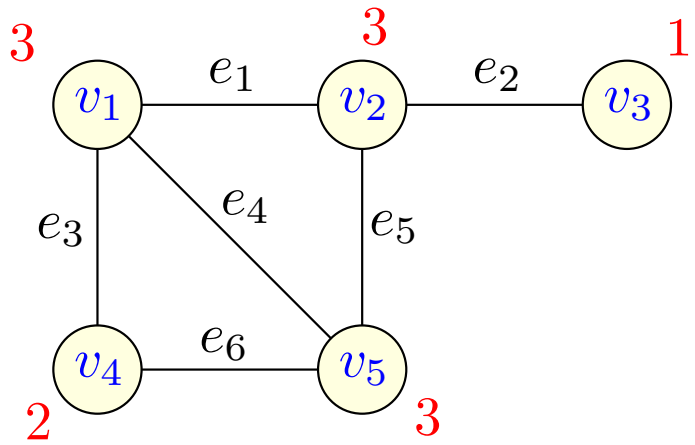


- Two vertices joined by an edge are called the *end vertices* (or *endpoints*) of the edge. Two vertices are *adjacent* if there is an edge whose end vertices are u and v. An edge is said to be *incident* on its endpoints.

Examples

v_1 and v_2 are the endpoints of e_1, so v_1 and v_2 are adjacent. On the other hand, v_1 and v_3 are not adjacent. Edge e_1 is incident to vertices v_1 and v_2.

Terminology

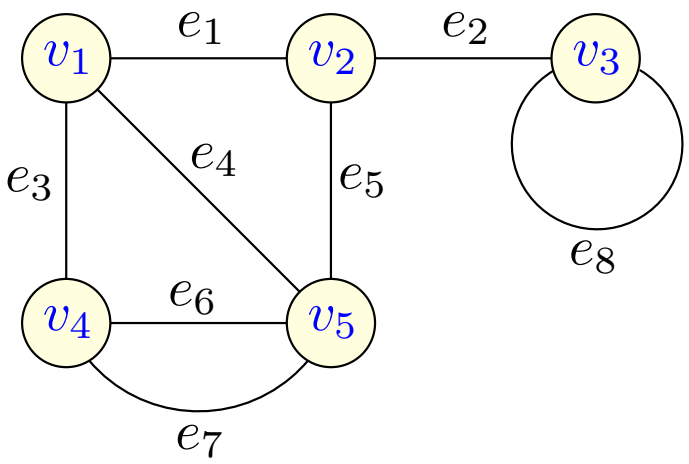


- The *degree* of a vertex is the number of edges incident to it.

Examples

$\deg(v_1) = \deg(v_2) = \deg(v_5) = 3$, $\deg(v_3) = 1$ and $\deg(v_4) = 2$.

Terminology

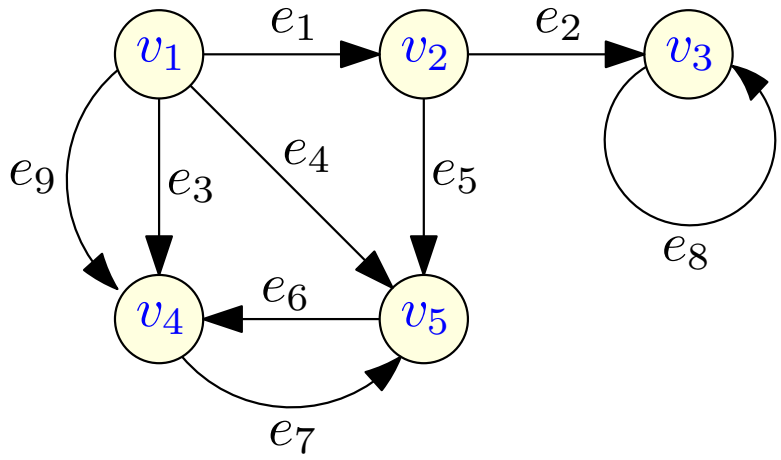


- Two edges may have the same endpoints, in which case they are called *multiple edges* or *parallel edges*. The two endpoints of an edge can be the same, in which case it is called *loop*.

Example

Edges e_6 and e_7 are parallel. We could also say that (v_4, v_5) is a multiple edge with multiplicity 2. The edge e_8 is a loop. We have $\deg(v_3) = 3$ and $\deg(v_4) = 3$.

Terminology

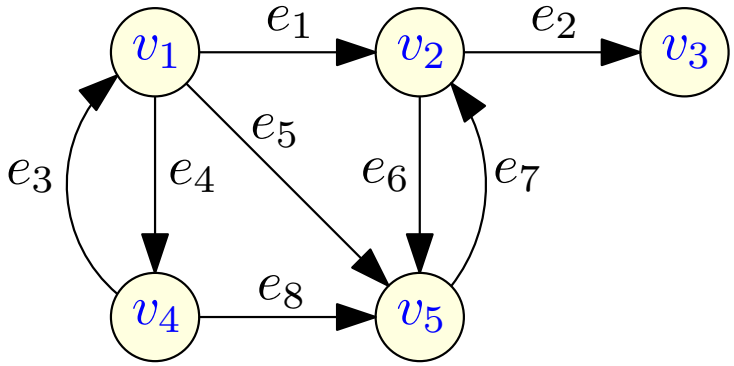


- In a directed graph, two edges are parallel if they have the same origin and the same destination.

Example

Edges e_3 and e_9 are parallel, but edges e_6 and e_7 are not parallel.

Terminology

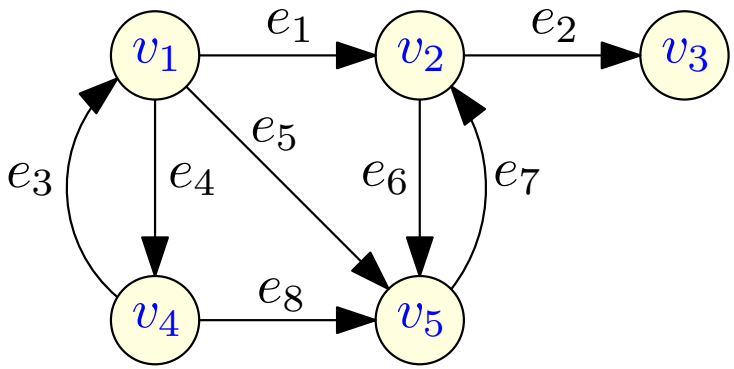


- The *outgoing edges* of a vertex are the directed edges whose origin is that vertex. The *incoming edges of a vertex* are the directed edges whose destination is that vertex.

Example

The outgoing edges of v_2 are e_2 and e_6. The incoming edges of v_2 are e_1 and e_7.

Terminology

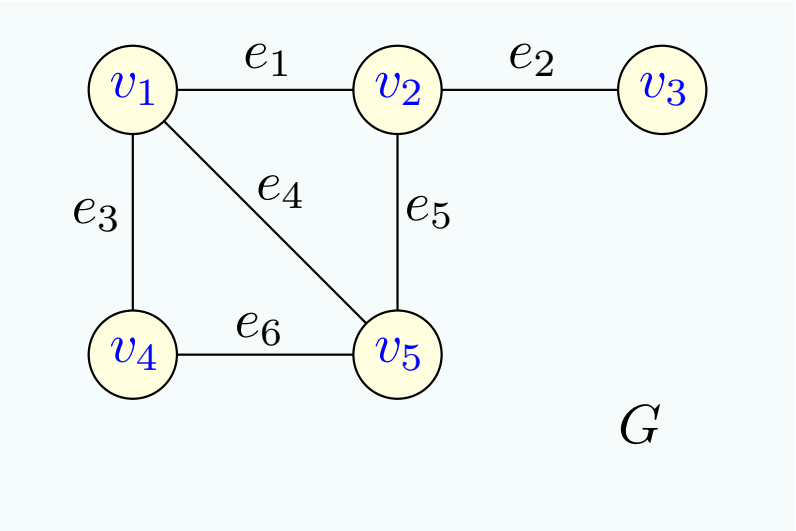


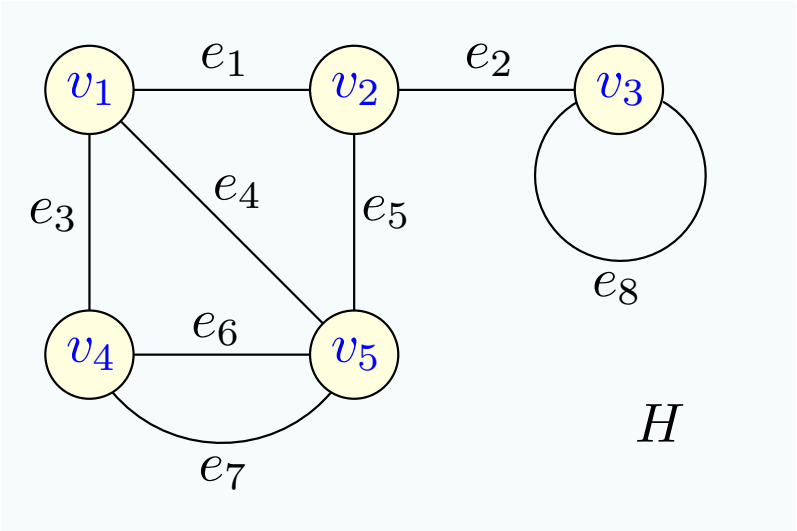
- The *in-degree* and *out-degree* of a vertex v are the number of the incoming and outgoing edges of v, and are denoted $\mathrm{indeg}(v)$ and $\mathrm{outdeg}(v)$, respectively

Example

$\mathrm{indeg}(v_1)$=1 and $\mathrm{outdeg}(v_1) = 3$.

Terminology





- Usually, graphs do not have any parallel edges or loops. Such graphs are said to be *simple graphs*.

Example

G is a simple graph and H is not.

Properties

Proposition

If $G = (V, E)$ is a graph with m edges, then

$$\sum_{v \in V} \deg(v) = 2m.$$

Proof.

Each edge (u, v) contributes 2 in this sum: One for $\deg(u)$ and one for $\deg(v)$. □

Proposition

If $G = (V, E)$ is a directed graph with m edges, then

$$\sum_{v \in V} \text{indeg}(v) = \sum_{v \in V} \text{outdeg}(v) = m.$$

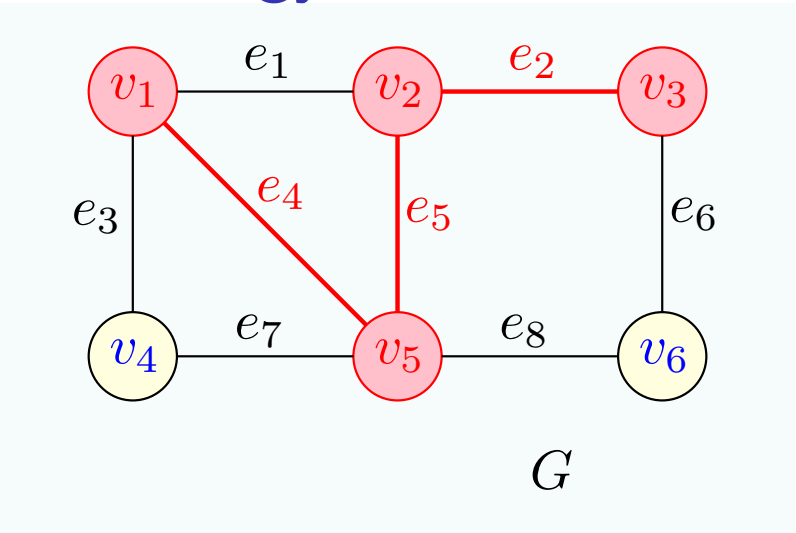
Properties

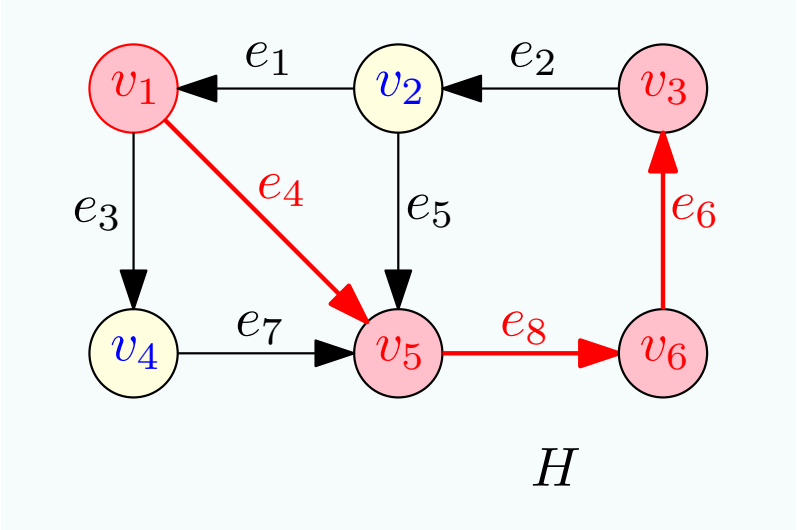
Proposition

Let G be a simple graph with n vertices and m edges. If G is undirected, then $m \leqslant n(n-1)/2$, and if G is directed, then $m \leqslant n(n-1)$.

- Proof done in class.

Terminology





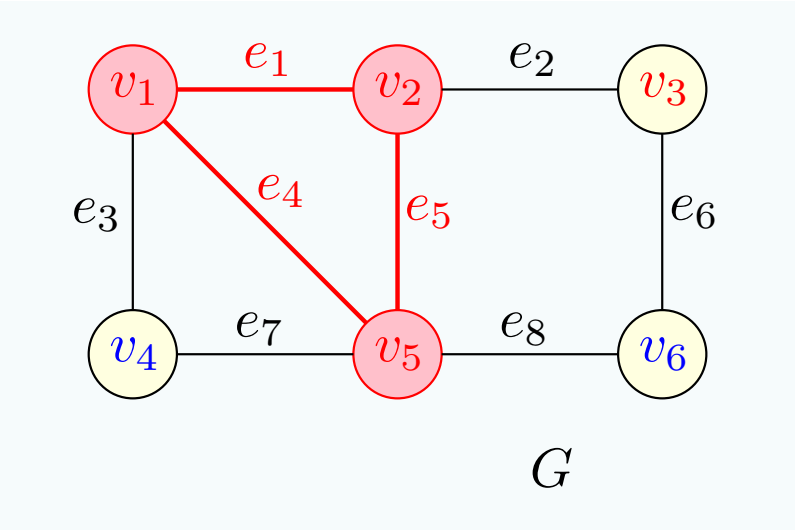
Definition

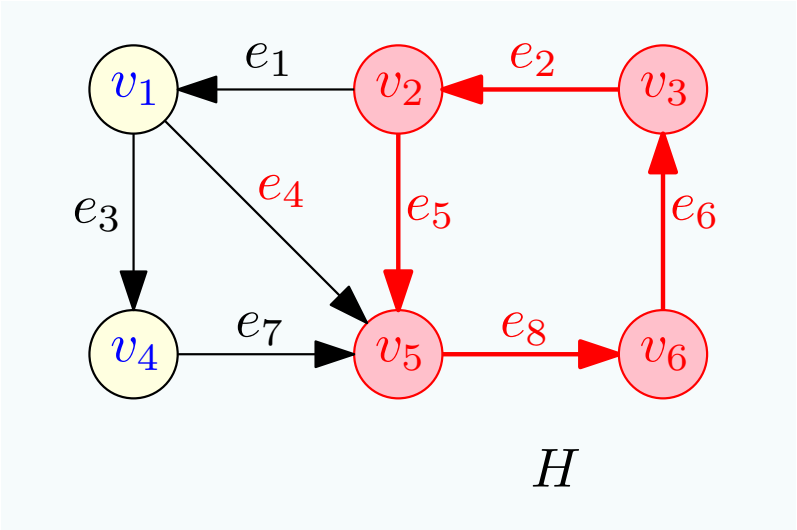
A *path* is a sequence of alternating vertices and edges that starts at a vertex and ends at a vertex such that each edge is incident to its predecessor and successor vertex. If the graph is directed, each edge must be traversed in its direction.

Example

(left) The path $(v_1, e_4, v_5, e_5, v_2, e_2, v_3)$ in the graph G.
(right) The directed path $(v_1, e_4, v_5, e_8, v_6, e_6, v_3)$ in the graph H.

Terminology





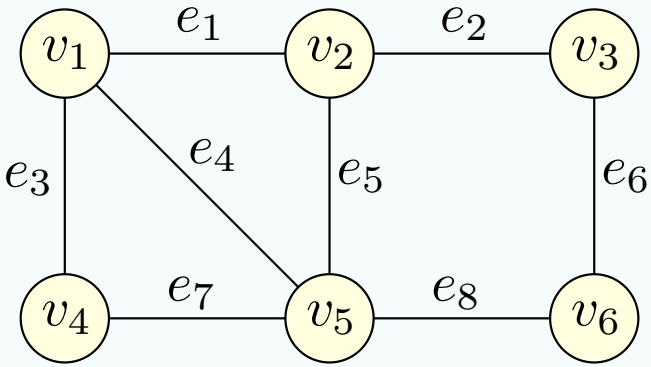
Definition

A *cycle* is a path that starts and ends at the same vertex.

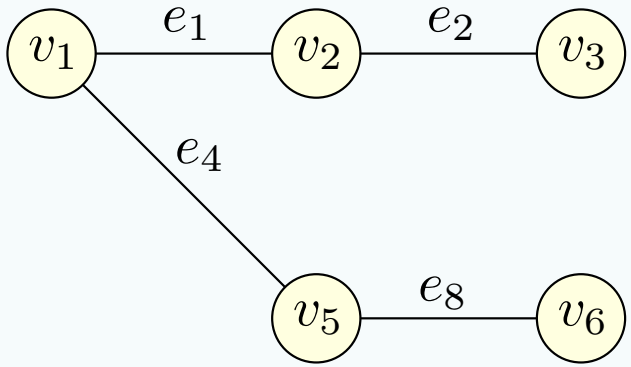
Example

(left) The cycle $(v_1, e_1, v_2, e_5, v_5, e_4, v_1)$ in the graph G.
(right) The directed cycle $(v_2, e_5, v_5, e_8, v_6, e_6, v_3, e_2, v_2)$.

Terminology



G

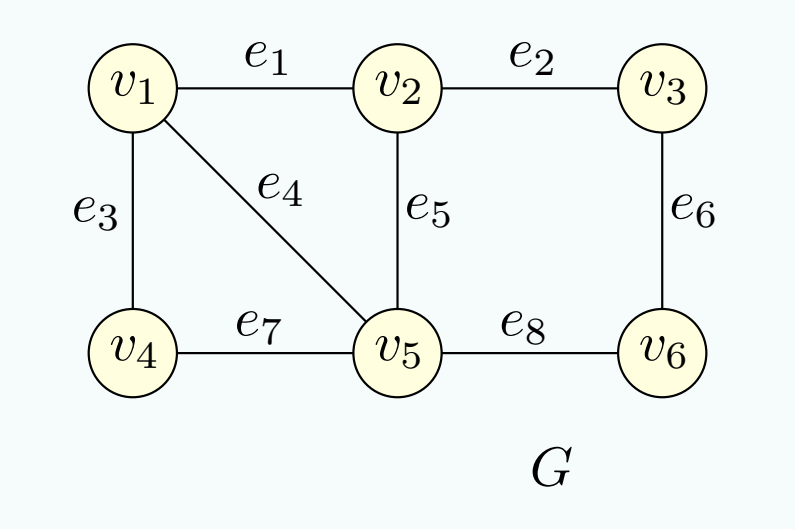


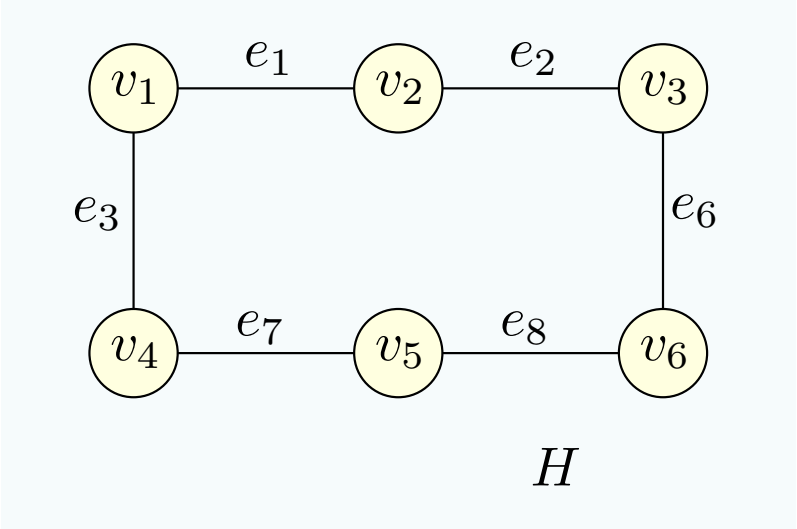
H

Definition

A *subgraph* of a graph $G = (E, V)$ is a graph $H = (F, W)$ such that $F \subseteq E$ and $W \subseteq V$.

Terminology

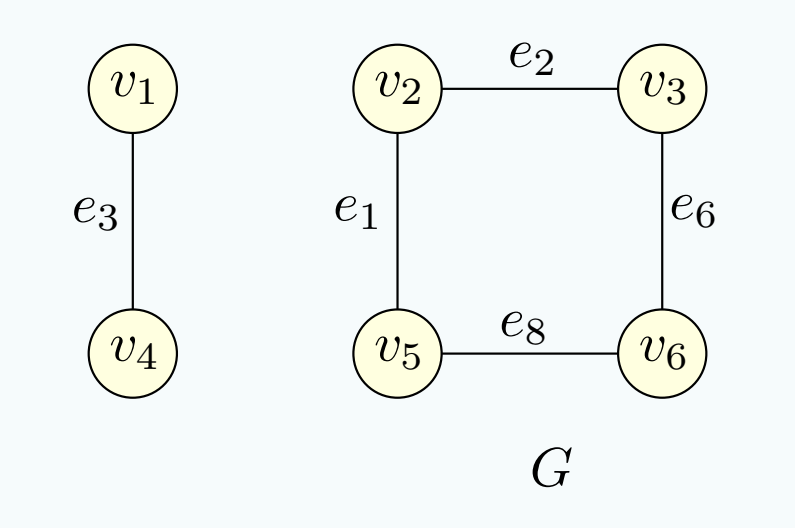


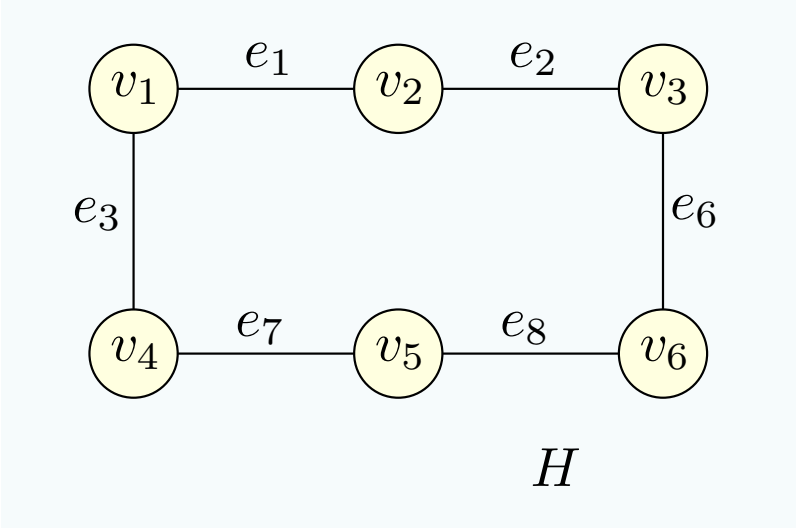


Definition

A *spanning subgraph* of a graph $G = (E, V)$ is a subgraph $H = (F, W)$ such that $W = V$.

Terminology





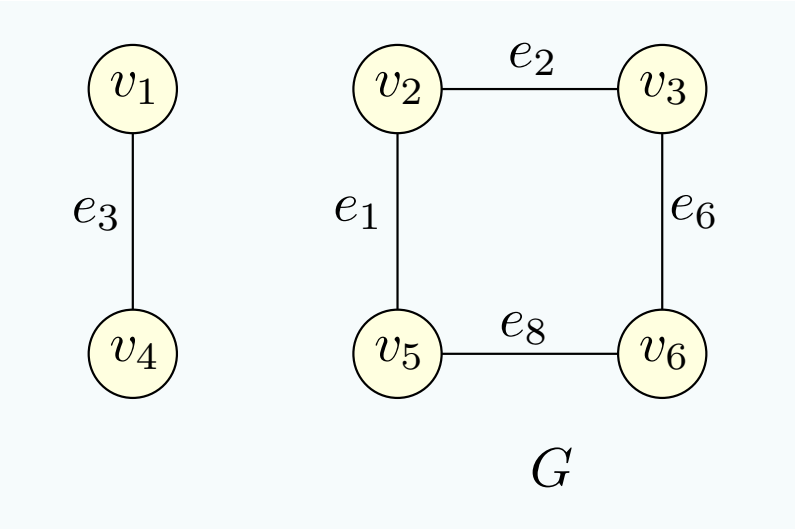
Definition

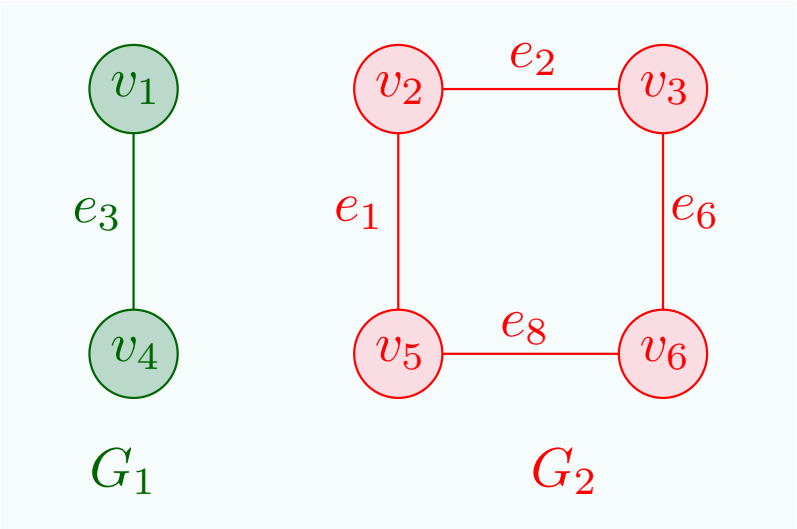
A graph is *connected* if there is a path between every two vertices.

Examples

G is disconnected because there is no path from v_1 to v_2.
H is connected.

Terminology





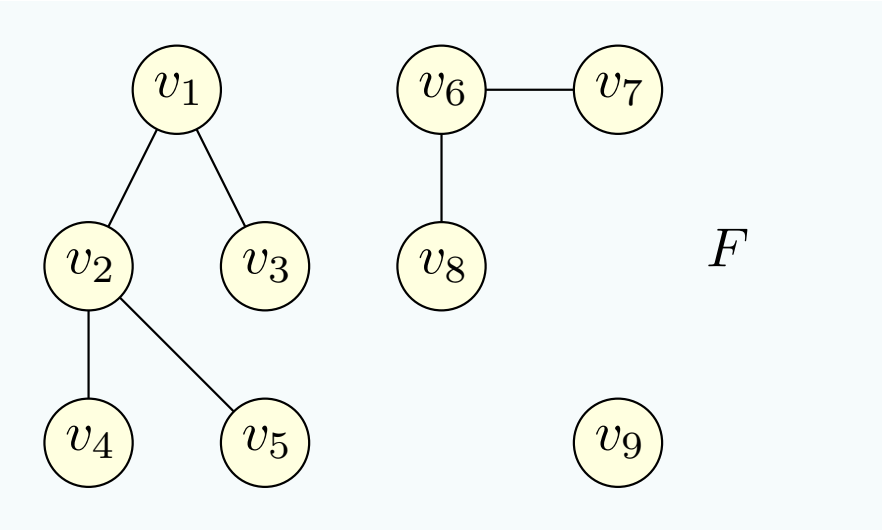
Definition

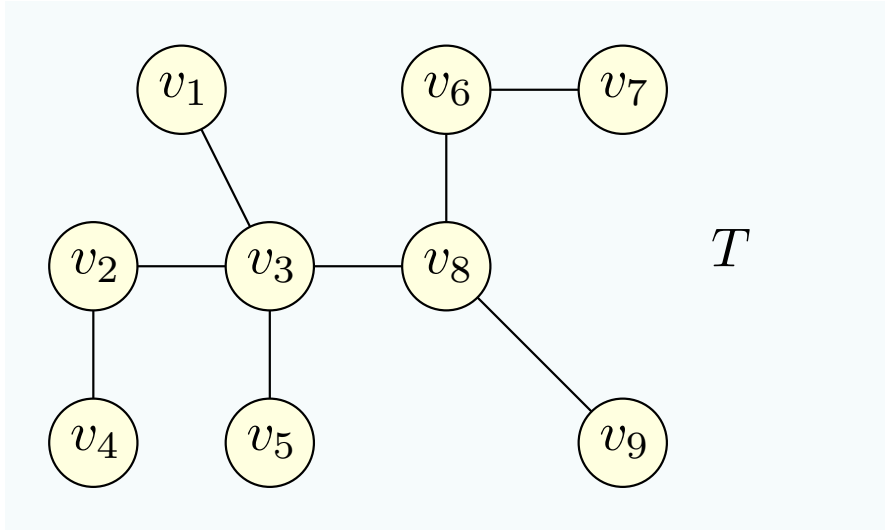
A *connected component* if a maximal connected subgraph

Example

G has two connected components, G_1 and G_2.

Terminology





Definitions

A *forest* is a graph without cycle. A *tree* is a connected forest.

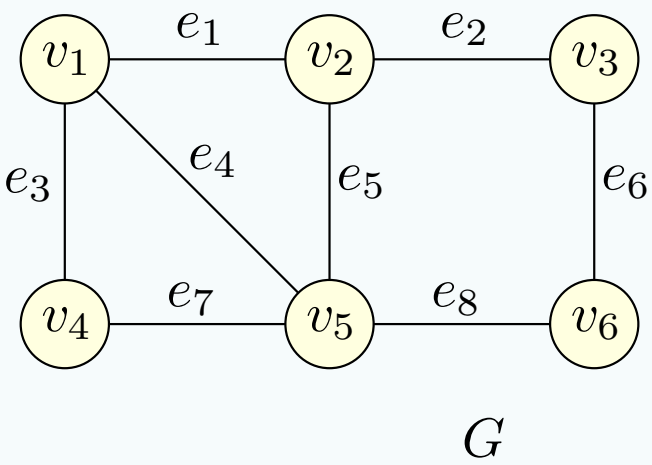
Examples

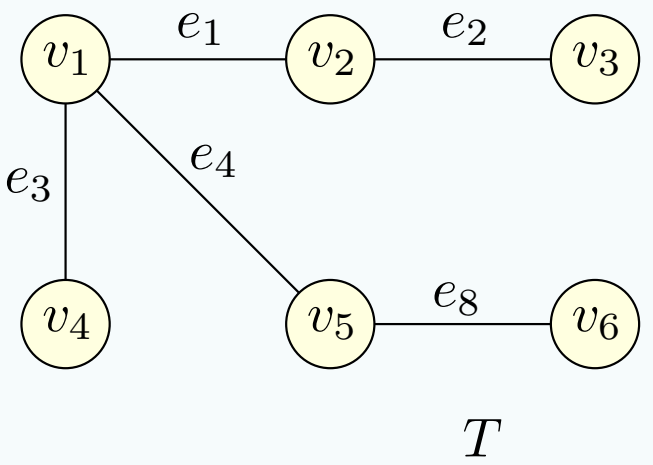
A forest F and a tree T.

- The trees presented here have no root: they are called *free trees*. The trees from Lecture 13 are called *rooted trees*.

Terminology

- The connected components of a forest are (free) trees.





Definition

A *spanning tree* T of a graph G is a spanning subgraph that is a tree.

Properties

Proposition

Let G be an undirected graph with n vertices and m edges.

- *If G is connected, then $m \geqslant n - 1$.*
- *If G is a tree, then $m = n - 1$.*
- *If G is a forest, then $m \leqslant n - 1$.*
- *If G is connected and $m = n - 1$, then G is a tree.*

The Graph ADT

- Elements are stored at vertices and edges. Each vertex object u supports the following operations:

- **operator***(): Return the element associated with u.
- **incidentEdges**(): Return an edge list of the edges incident on u.
- **isAdjacentTo**(v): Test whether vertices u and v are adjacent.

- Each Edge object e supports the following operations:

- **operator***(): Return the element associated with e.
- **endVertices**(): Return a vertex list containing e's end vertices.
- **opposite**(v): Return the end vertex of edge e distinct from vertex v.
- **isAdjacentTo**(f): Test whether edges e and f are adjacent.
- **isIncidentOn**(v): Test whether e is incident on v.

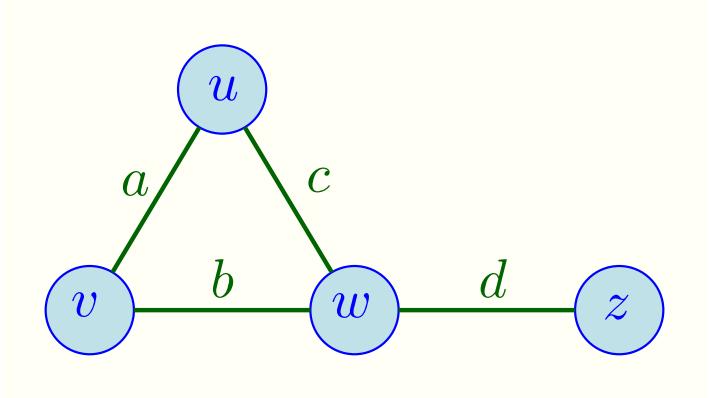
The Graph ADT

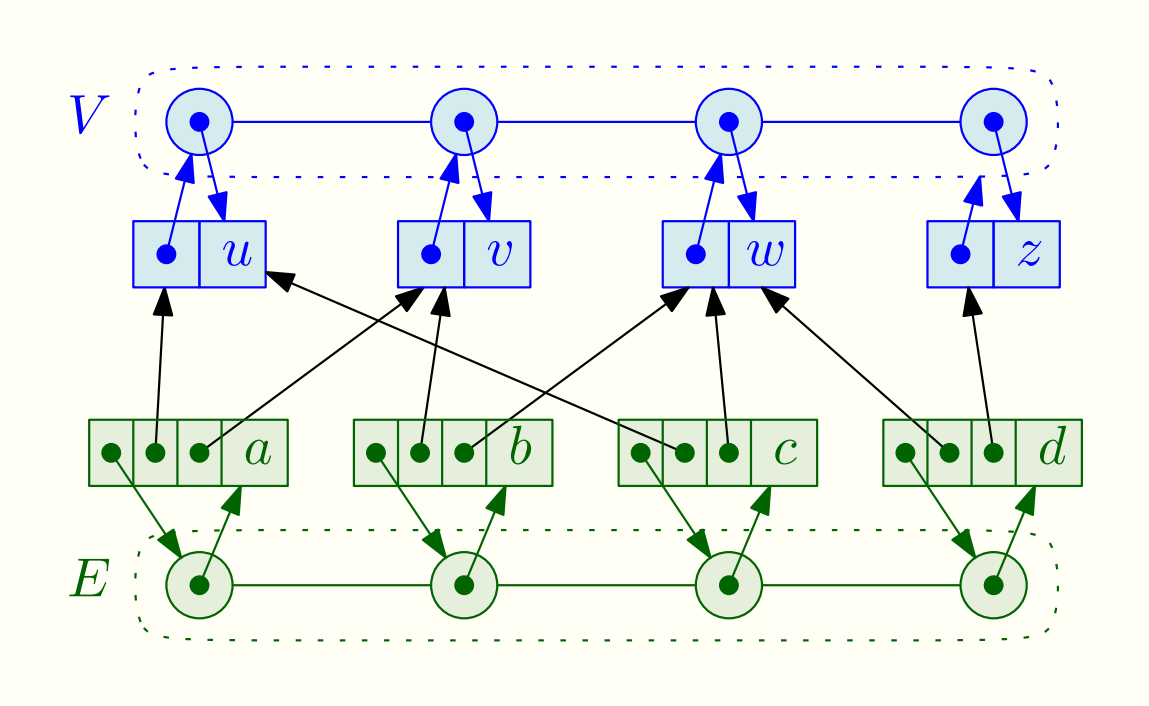
- Finally, the full graph ADT consists of the following operations:

- **vertices**(): Return a vertex list of all the vertices of the graph.
- **edges**(): Return an edge list of all the edges of the graph.
- **insertVertex**(x): Insert and return a new vertex storing element x.
- **insertEdge**(v, w, x): Insert and return a new undirected edge with end vertices v and w and storing element x.
- **eraseVertex**(v): Remove vertex v and all its incident edges.
- **eraseEdge**(e): Remove edge e.

- The VertexList and EdgeList classes support the standard list operations. In particular, we assume that each provides an iterator (Lecture 11), which we call VertexItor and EdgeItor, respectively.

The Edge List Structure





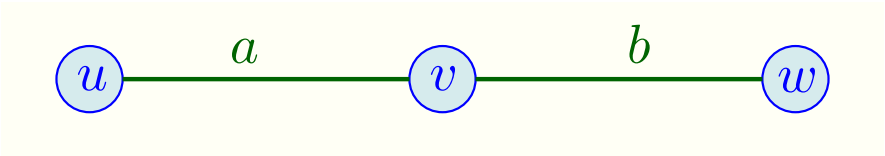
The Edge List Structure

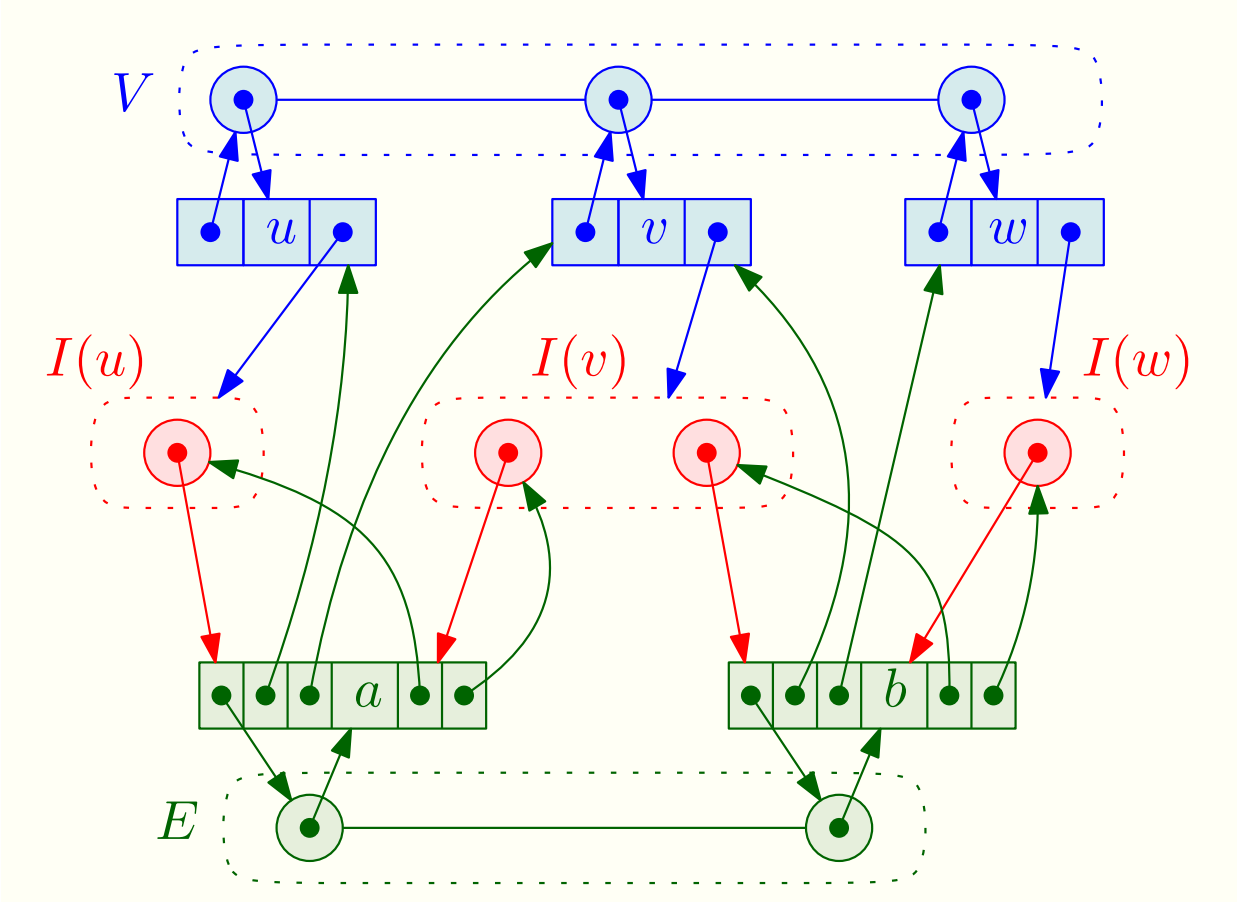
- This structure is called the *edge list* because E can simply be stored in a list.
- But we may also store it in a dictionary (where the key is the element and the edge is the value) so as to be able to search for specific objects associated with edges.
- We can also store V in a dictionary.

Performance of The Edge List Structure

Operation	Time
vertices	$O(n)$
edges	$O(m)$
endVertices, opposite	$O(1)$
incidentEdges, isAdjacentTo	$O(m)$
isIncidentOn	$O(1)$
insertVertex, insertEdge, eraseEdge	$O(1)$
eraseVertex	$O(m)$

The Adjacency List Structure





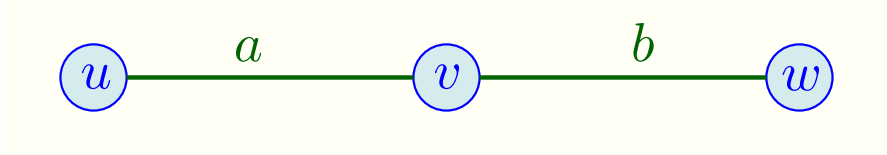
The Adjacency List Structure

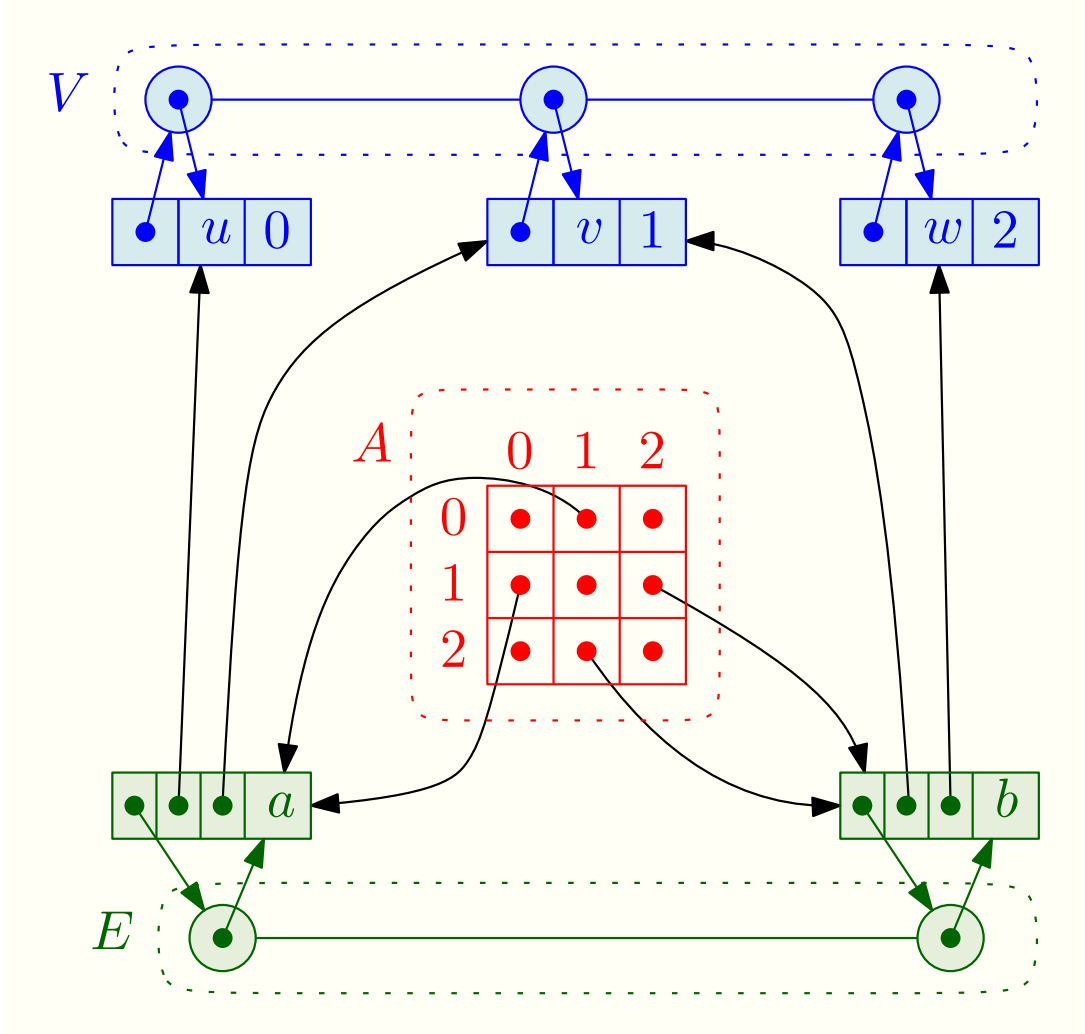
- The *adjacency list* structure includes all the structural components of the edge list structure plus the following:
  - A vertex object v holds a reference to a collection $I(v)$, called the *incidence collection* of v, whose elements store references to the edges incident on v.
  - The edge object for an edge e with end vertices v and w holds references to the positions (or entries) associated with edge e in the incidence collections $I(v)$ and $I(w)$.

Performance of Adjacency List Structure

Operation	Time
vertices	$O(n)$
edges	$O(m)$
endVertices, opposite	$O(1)$
v.incidentEdges	$O(\deg(v))$
v.isAdjacentTo(w)	$O(\min(\deg(v), \deg(w)))$
isIncidentOn	$O(1)$
insertVertex, insertEdge, eraseEdge	$O(1)$
eraseVertex(v)	$O(\deg(v))$

The Adjacency Matrix Structure





The Adjacency Matrix Structure

- The *adjacency matrix* structure extends the edge list structure as follows:
  - A vertex object v stores a distinct integer i in the range $0, 1, \ldots, n-1$, called the index of v.
  - We keep a two-dimensional $n \times n$ array A such that the cell $A[i, j]$ holds a reference to the edge (v, w), if it exists, where v is the vertex with index i and w is the vertex with index j. If there is no such edge, then $A[i, j] = \text{null}$.

Performance of the Adjacency Matrix Structure

Operation	Time
vertices	$O(n)$
edges	$O(m)$
endVertices, opposite	$O(1)$
isAdjacentTo, isIncidentOn	$O(1)$
incidentEdges	$O(n)$
insertEdge, eraseEdge	$O(1)$
insertVertex, eraseVertex	$O(n^2)$