14 GRAPH PROPERTIES

GRAPHS – PROPERTIES

Proposition: If G is a graph with m edges, then:

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

Justification:

- An edge (u,v) is counted twice in the summation above:
  - once by its endpoint u when considered an origin
  - once by its endpoint v when considered an origin
- Thus, the total contribution of the edges to the degrees of the vertices is twice the number of edges.

GRAPHS – PROPERTIES

Proposition: If G is a **directed** graph with m edges, then:

$$\sum_{v \in G} indeg(v) = \sum_{v \in G} outdeg(v) = m$$

Justification:

- In a **directed** graph, an edge (u,v) contributes:
  - One unit to the **out-degree** of its origin u
  - One unit to the **in-degree** of its destination v
- Based on this:
  - The total contribution of the edges to the **in-degrees** equals the number of edges
  - The total contribution of the edges to the **out-degrees** equals the number of edges

GRAPHS – PROPERTIES

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

Justification:

- If G is a simple **undirected** graph:
  - Since G is simple, then for each vertex v we have: $deg(v) \leq n-1$, implying that:

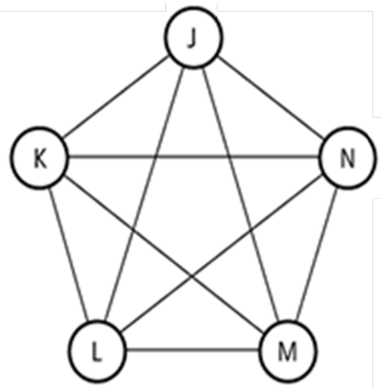
$$\sum_{v \in G} deg(v) \leq n(n-1)$$

  - Since G is **undirected**, then we already proved that

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

  - The above two equations imply that:

$$2m \leq n(n-1) \equiv m \leq n(n-1)/2$$



Example: Given a simple undirected graph of **5** vertices, the **maximum** possible degree of a vertex is **4**

GRAPHS – PROPERTIES

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

Justification:

- If G is a simple **directed** graph:
  - Since G is simple, for each vertex v we have: $indeg(v) \leq n-1$, implying that:

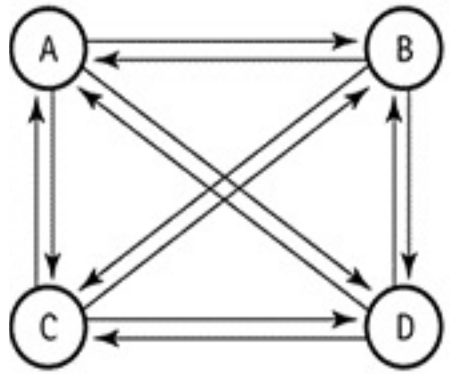
$$\sum_{v \in G} indeg(v) \leq n(n-1)$$

  - Since G is **directed**, then we already proved that

$$\sum_{v \in G} indeg(v) = m$$

  - The above two equations imply that:

$$\mathrm{m} \leq n(n-1)$$



Example: Given a simple directed graph of **4** vertices, the **maximum** possible in-degree of a vertex is **3**

THE GRAPH ADT

- The abstract data type has a **position** for each vertex, and a **position** for each edge.
- Each **Vertex** object, *u*, supports at least the following operations:

  operator*(): Return the **element** associated with *u*.

  incidentEdges(): Return an **edge list** of all the edges incident on *u*.

  isAdjacentTo(*v*): Test whether vertices *u* and *v* are **adjacent**.

- Each **Edge** object *e* supports at least the following operations:

  operator*(): Return the **element** associated with *e* (e.g., it could be the weight of *e*)

  endVertices(): Return a **vertex list** containing *e*'s **end vertices**.

  opposite(*v*): Return the **end vertex** of edge *e* distinct from vertex *v* (an error occurs if *e* is not incident on v).

  isAdjacentTo(*f*): Test whether edges *e* and *f* are **adjacent**.

  isIncidentOn(*v*): Test whether *e* is **incident** on *v*.

THE GRAPH ADT

- The **Graph ADT** itself supports at least 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*,*z*): **Insert** and return a new **undirected edge** with end vertices *v* and *w* and storing element *z*.

  eraseVertex(*v*): **Remove vertex** *v* and all its incident edges.

  eraseEdge(*e*): **Remove edge** *e*.

21

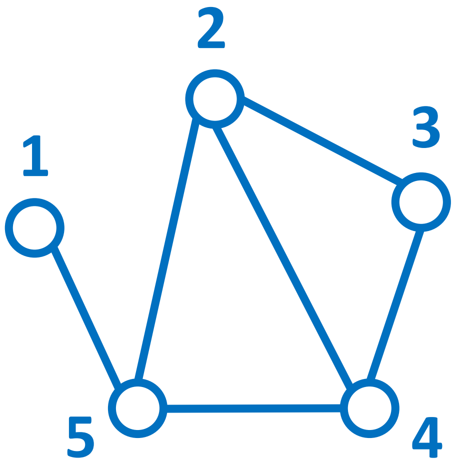
DATA STRUCTURES FOR GRAPHS

DATA STRUCTURES FOR GRAPHS

How would you represent such a graph in memory?

We can use an EDGE LIST, which simply lists all the edges one by one:

- (1, 5)
- (2, 3)
- (2, 4)
- (2, 5)
- (3, 4)
- (4, 5)



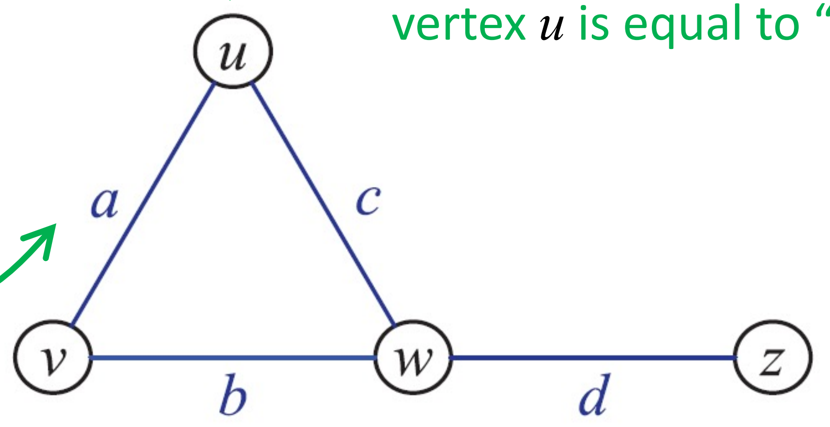
EDGE LIST

Here is the edge list of this graph:

- (v, u)
- (v, w)
- (u, w)
- (w, z)

For simplicity, assume that the element associated with vertex u is equal to "u"

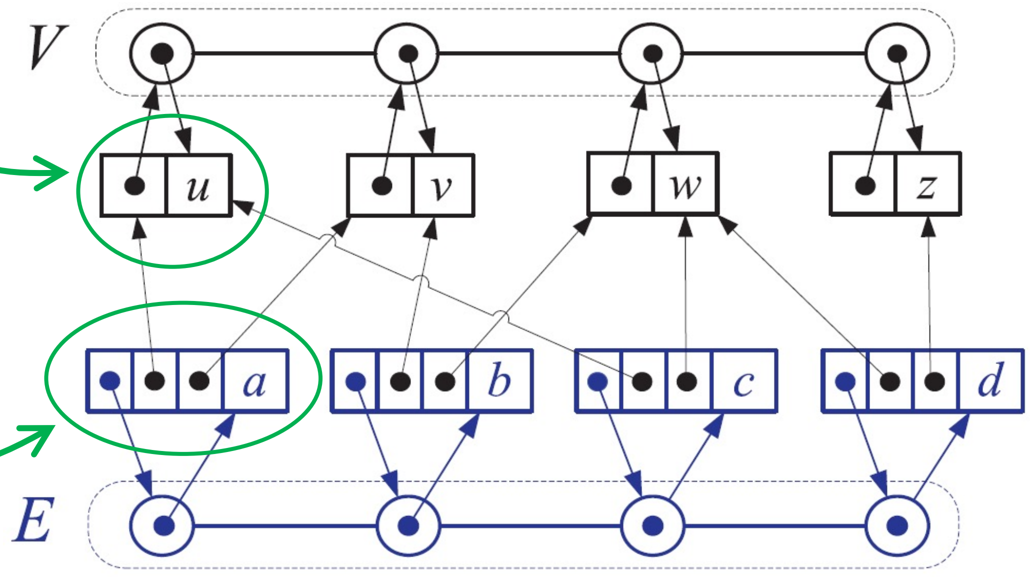
a is the element associated with (u, v) (e.g., it represents the edge's weight)



Here is how to represent this edge list:

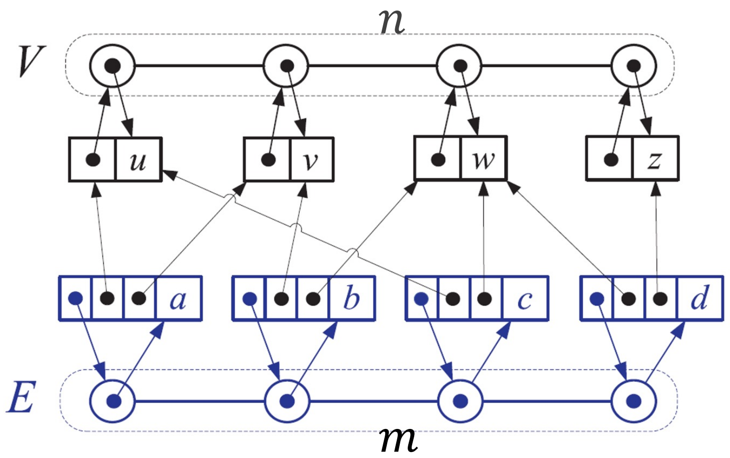
A vertex object consists of the **element** u and the **position** of the object in the collection V

An edge object consists of the **element** a and the object's **position** in the collection E as well as **positions** associated with its two endpoints, u and v



EDGE LIST – COMPLEXITY

- Given an **Edge List**, what is the complexity of these operations?



$O(n)$ vertices: Return a list of **all vertices**

$O(m)$ edges: Return a list of **all edges**

$O(1)$ insertVertex: Inserts a **new vertex**

$O(1)$ insertEdge: Insert a **new edge**

$O(m)$ eraseVertex: **Remove a vertex** and all its incident edges

$O(1)$ eraseEdge: **Remove an edge**

$O(m)$ incidentEdges: Return a list of **all edges incident** on a vertex

$O(m)$ isAdjacentTo: Test whether **two vertices are adjacent**

$O(1)$ endVertices: Return the two **endpoints** of an edge

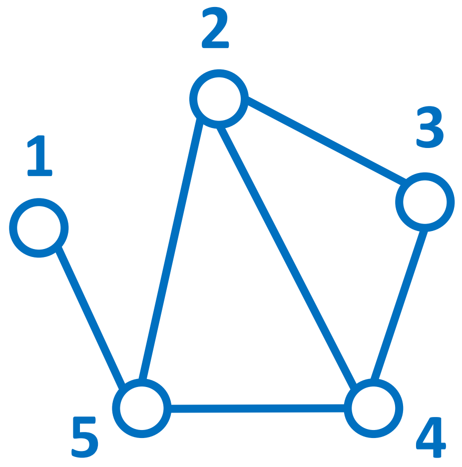
$O(1)$ opposite: Given an endpoint of an edge, **return the other endpoint**

$O(1)$ isIncidentOn: Test whether **an edge is incident** on given vertex

ADJACENCY LIST

We discussed how this graph can be represented as an edge list, which simply **lists the edges**:

- (1, 5)
- (2, 3)
- (2, 4)
- (2, 5)
- (3, 4)
- (4, 5)



An alternative is an ADJACENCY LIST, which **lists the vertices that are adjacent to each vertex**

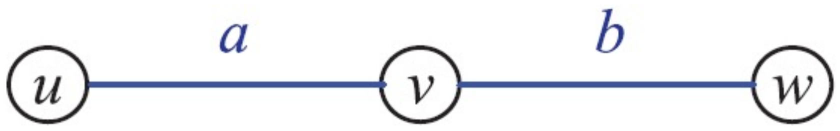
- **1:** 5
- **2:** 3, 4, 5
- **3:** 2, 4,
- **4:** 2, 3, 5
- **5:** 1, 2, 4

This is called the "incidence collection" of vertex **2**, denoted by $\mathrm{I}(2)$

ADJACENCY LIST

Here is the adjacency list of this graph:

- u: v
- v: u, w
- w: v

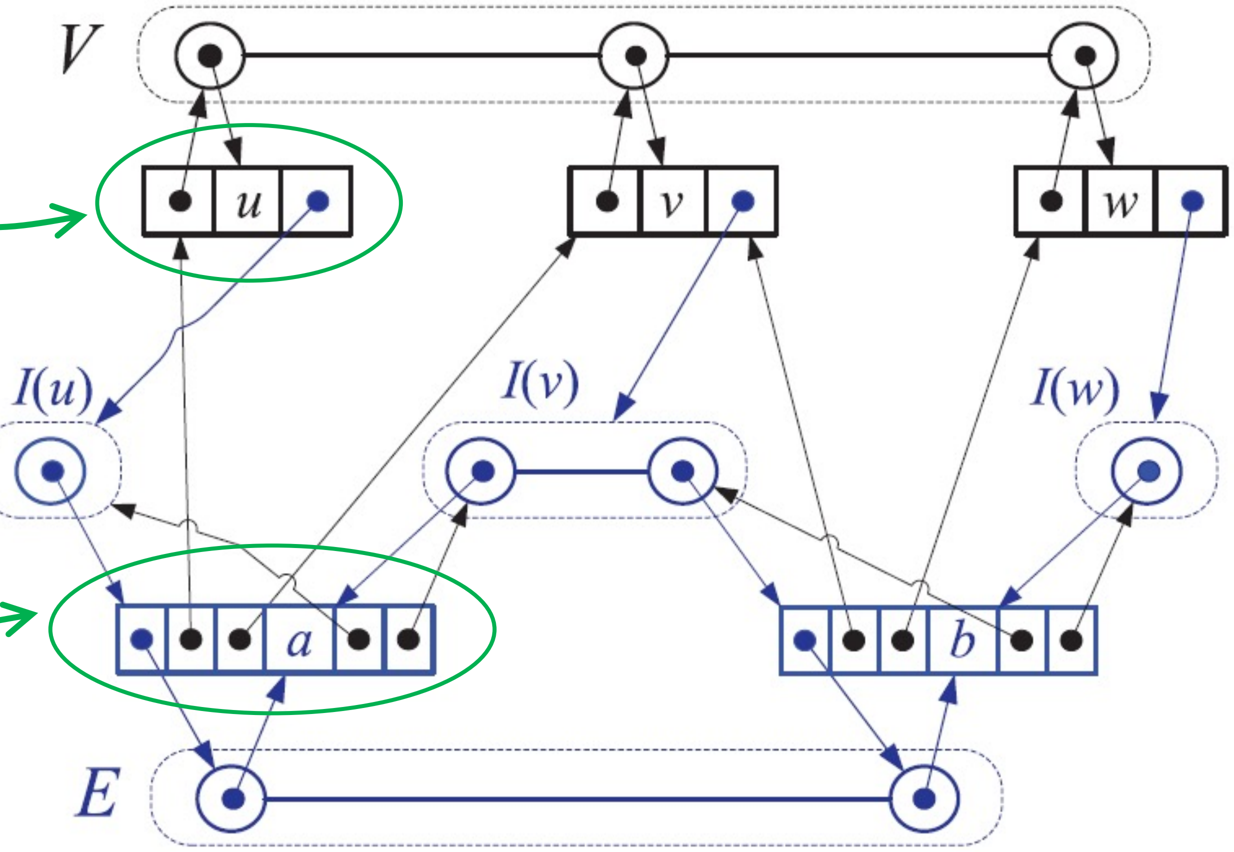


A vertex object consists of:

- The **element** u
- The object's **position** in collection V
- A **reference** to $\mathrm{I}(u)$

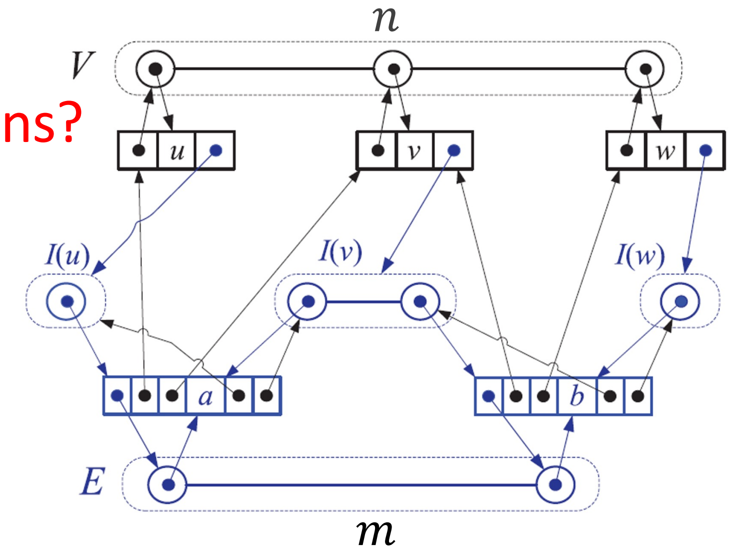
An edge object consists of:

- The **element** a
- The object's **position** in the collection E
- The **positions** associated with u and v
- **References** to the edge's position in $\mathrm{I}(u)$ and the edge's position in $\mathrm{I}(v)$



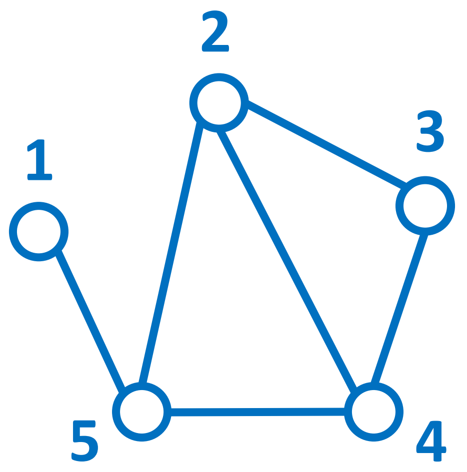
ADJACENCY LIST – COMPLEXITY

- Given an **adjacency list**, what is the complexity of these operations?



$O(n)$	vertices: Return a list of **all vertices**
$O(m)$	edges: Return a list of **all edges**
$O(1)$	insertVertex: Inserts a **new vertex**
$O(1)$	insertEdge: Insert a **new edge**
$O(\deg(v))$	eraseVertex(v): **Remove a vertex** and all its incident edges
$O(1)$	eraseEdge: **Remove an edge**
$O(\deg(v))$	v.incidentEdges(): Return a list of **all edges incident** on a vertex
$O(\min(\deg(v), \deg(w)))$	v.isAdjacentTo(w): Test whether v and w are **adjacent**
$O(1)$	endVertices: Return the two **endpoints** of an edge
$O(1)$	opposite: Given an endpoint of an edge, **return the other endpoint**
$O(1)$	isIncidentOn: Test whether **an edge is incident** on given vertex

ADJACENCY MATRIX



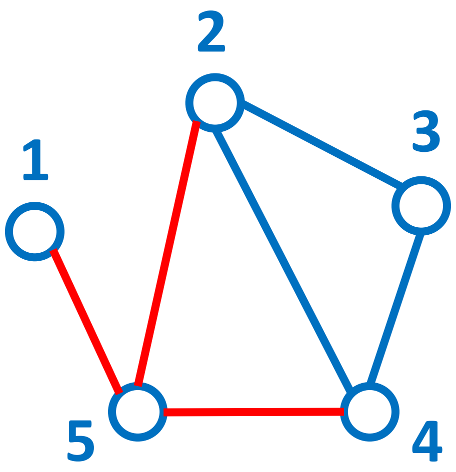
An edge list simply lists **the edges**:

- (1, 5)
- (2, 3)
- (2, 4)
- (2, 5)
- (3, 4)
- (4, 5)

An adjacency list specifies the **vertices that are adjacent to each vertex**:

- **1:** 5
- **2:** 3, 4, 5
- **3:** 2, 4,
- **4:** 2, 3, 5
- **5:** 1, 2, 4

ADJACENCY MATRIX



An edge list simply lists **the edges**:

- (1, 5)
- (2, 3)
- (2, 4)
- (2, 5)
- (3, 4)
- (4, 5)

An adjacency list specifies the **vertices that are adjacent to each vertex**:

- **1:** 5
- **2:** 3, 4, 5
- **3:** 2, 4,
- **4:** 2, 3, 5
- **5:** 1, 2, 4

We can use an ADJACENCY MATRIX, where:

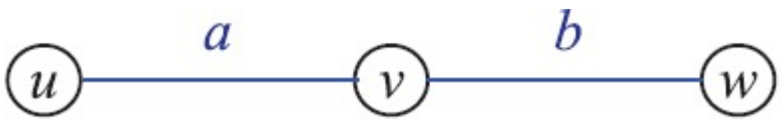
- $A[i, j]$ = 1 means j is adjacent to i
- $A[i, j]$ = 0 means j is not adjacent to i

$$\begin{array}{c c} & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

ADJACENCY MATRIX

$A[i, j]$ holds a **reference** to the edge between the vertices whose indices are i and j (if such an edge exists)

Here is the adjacency Matrix of this graph:



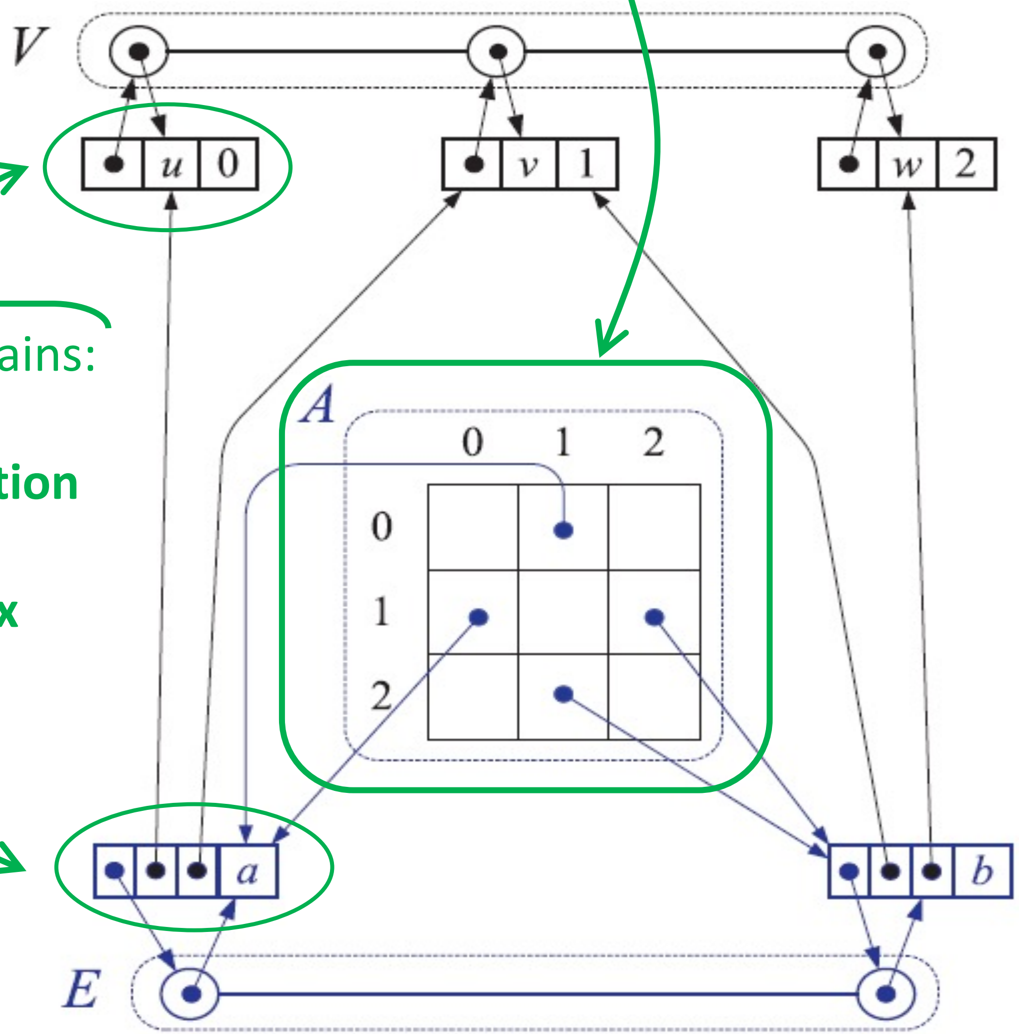
$$\begin{array}{c} \\ u \\ v \\ w \end{array}\begin{array}{c} \begin{array}{ccc} u & v & w \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \end{array}$$

A vertex object contains:

- The **element** u
- The object's **position** in collection V
- The vertex's **index**

An edge object consists of:

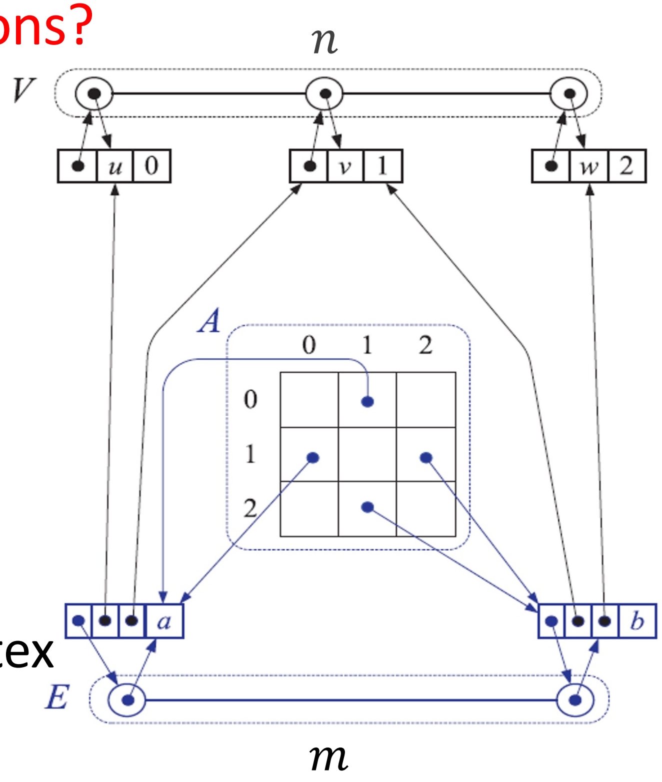
- The **element** a
- The object's **position** in the collection E
- The **positions** associated with the endpoints u and v



ADJACENCY MATRIX – COMPLEXITY

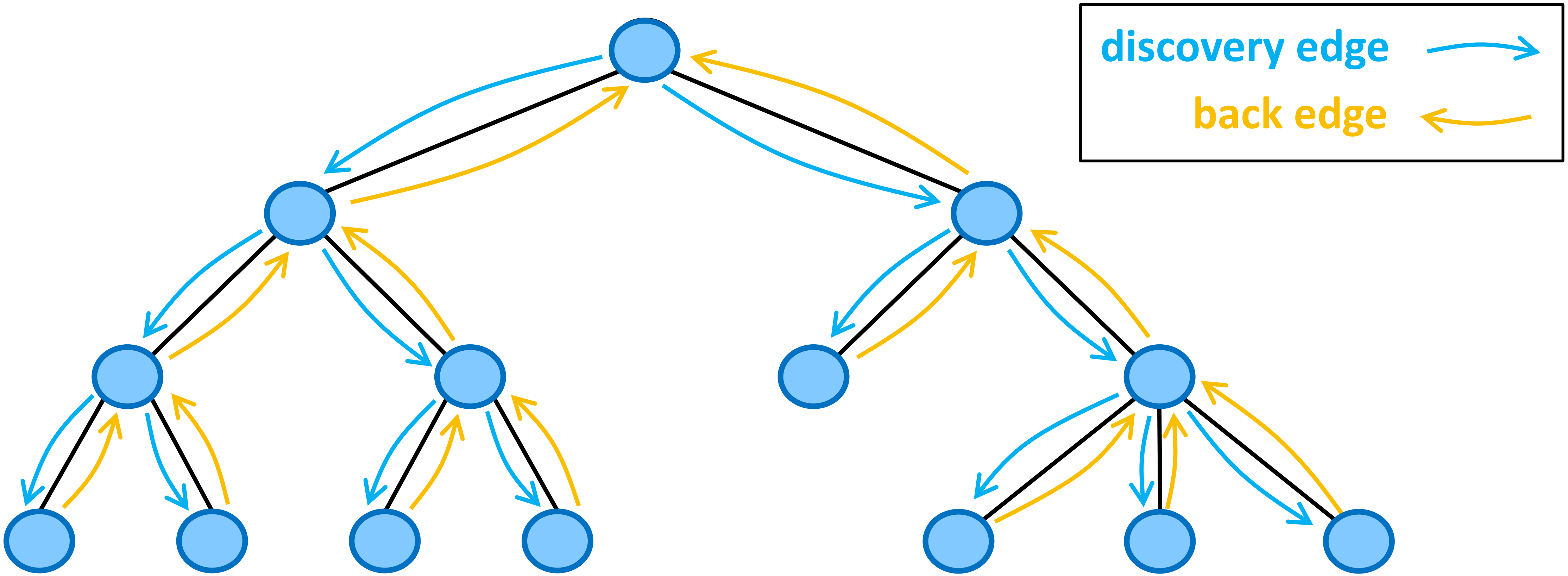
- Given an **adjacency matrix**, what is the complexity of these operations?

$O(n)$	vertices: Return a list of **all vertices**
$O(n^2)$	edges: Return a list of **all edges**
$O(n^2)$	insertVertex: Inserts a **new vertex**
$O(1)$	insertEdge: Insert a **new edge**
$O(n^2)$	eraseVertex: **Remove a vertex** and all its incident edges
$O(1)$	eraseEdge: **Remove an edge**
$O(n)$	incidentEdges: Return a list of **all edges incident** on a vertex
$O(1)$	isAdjacentTo: Test whether **two vertices are adjacent**
$O(1)$	endVertices: Return the two **endpoints** of an edge
$O(1)$	opposite: Given an endpoints of an edge, **return the other endpoint**
$O(1)$	isIncidentOn: Test whether **an edge is incident** on given vertex



DEPTH-FIRST SEARCH (DFS)

- How to traverse a **tree**, i.e., visit all of its vertices one by one?
- One way to do this is to use the "depth-first search" algorithm:



BREADTH-FIRST SEARCH (BFS)

- An alternative to **depth-first search** is to called “breadth-first search”, which works in trees as follows: