# Computer Science 331

## Graphs and Their Representations


Mike Jacobson

Department of Computer Science
University of Calgary


Lecture #29



## Outline







## Undirected Graphs

An *undirected graph* $G = (V, E)$ consists of

- a finite, nonempty set $V$ of *vertices* or "nodes"
- a set $E$ of *edges*, where each "edge" is an unordered pair of distinct elements of $V$

Also may be written as $V(G)$ and $E(G)$ to indicate association to a particular graph.

Undirected graphs, and their generalizations, can be used to model

- communication networks
- knowledge and data bases

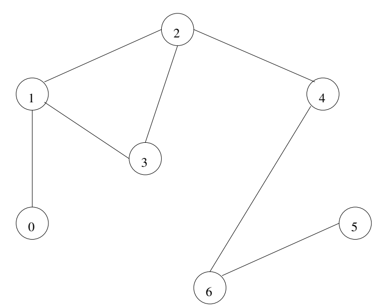
Graphs and their algorithms will be studied for the rest of this course.





## Example

$G$:




$G = (V, E)$ where

- $V = \{0, 1, 2, 3, 4, 5, 6\}$
- $E = \{(0, 1), (1, 2), (1, 3), (2, 3), (2, 4), (4, 6), (5, 6)\}$

## Terminology

If $u, v \in V$ and $u \neq v$ then $u$ and $v$ are **neighbours** (or, "$u$ is **adjacent** to $v$") if $(u, v) \in E$.

If $u \in V$ then the **degree** of $u$ is the number of neighbours of $u$.

Note that if $|V| = n$ then $|E| \leq \binom{n}{2} = \frac{n(n-1)}{2}$.

- The graph $G = (V, E)$ is **dense** if $|E| \in \Omega(n^2)$ (for $n = |V|$)
- The graph $G = (V, E)$ is **sparse** if $|E|$ is significantly smaller than $n^2$.





## Operations

The following operations should be supported:

- **Creation:** It should be possible to
  - initialize a graph to be empty (with no vertices or edges),
  - add another vertex
  - add an edge (between a pair of existing vertices that are not already neighbours);
- **Queries:** It should be possible to
  - ask whether a given pair of vertices are neighbours,
  - determine the number of vertices,
  - determine the number of edges;
- **Iterate:** It should be possible to iterate over
  - the set of vertices in the graph, as well as
  - the set of neighbours of any given vertex.





## Adjacency-Matrix Representation

**Assumption:** Vertices are numbered $0, 1, \dots, |V| - 1$ in some way.

The *adjacency-matrix* representation of $G$ consists of a $|V| \times |V|$ matrix $A_G$, with $(i, j)^{\text{th}}$ entry $a_{i,j}$ for $0 \leq i, j < |V|$, where

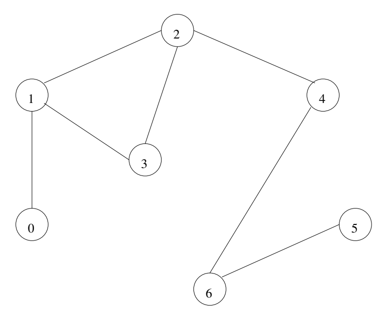
$$a_{i,j} = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{if } (i, j) \notin E. \end{cases}$$





## Example

$G$:




$A_G$:

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

**Note:** $A_G$ is a **symmetric** matrix: $a_{i,j} = a_{j,i}$ for $0 \leq i, j < |V|$.

## Properties

**Properties of This Representation:**

- simple
- reasonably space-efficient if $G$ is **dense**
- **not** space-efficient if $G$ is sparse!
- possible to add an edge or determine whether two vertices are neighbours in constant time
- iterating over the set of neighbours of a vertex requires $\Theta(|V|)$ operations, even if $G$ is sparse

. . . a good choice if $G$ is small or dense, not if large and sparse





## Adjacency-List Representation

The **adjacency-list** representation of $G = (V, E)$ consists of an array $Adj_G$ of $|V|$ lists, one for each vertex in $V$.

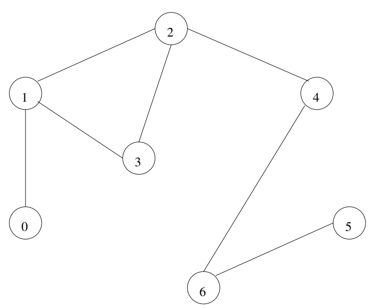
For each $u \in V$, the adjacency list $Adj_G(u)$ contains (pointers to) all the vertices $v \in V$ such that $(u, v) \in E$.



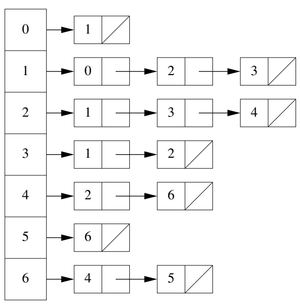


## Example

$G$:




$Adj_G$:








## Properties

**Properties of This Representation:**

- space-efficient if $G$ is **sparse**
- not really space-efficient if $G$ is (extremely) dense!
- checking whether a pair of vertices are neighbours requires more than constant time — number of operations is linear in the degree of one of the inputs, in the worst case
- adding an edge also requires this cost (if error checking is to be included)
- iterating over the set of neighbours of a vertex is efficient: Number of operations used is linear in the degree of the input vertex

. . . a good choice if $G$ is large and sparse; not if small or dense

## Directed Graphs

A *directed graph* ("digraph") $G = (V, E)$ consists of

- a finite, nonempty set $V$ of vertices or nodes, and
- a set $E$ of **ordered** pairs of elements of $E$ (that are not necessarily distinct)

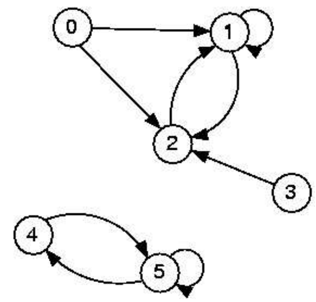
Directed graphs can be represented using adjacency-matrices or adjacency-lists, in much the same way that undirected graphs can.





## Example

$G$:




Adjacency-Matrix:

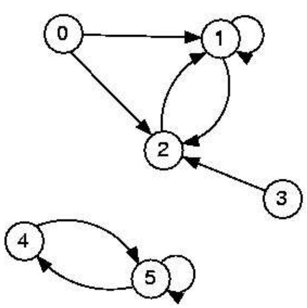
$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$



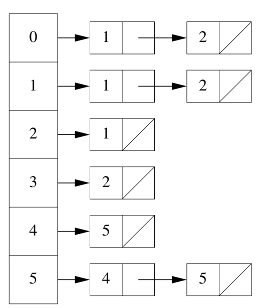


## Example

$G$:




Adjacency-List:








## Weighted Graphs

A *weighted* graph is an undirected or directed graph $G = (V, E)$ for which each *edge* has an associated **weight**.

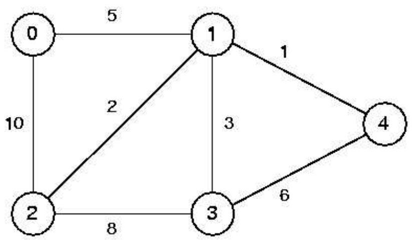
The weights are typically given an associated *weight function*

$$w : E \rightarrow \mathbb{R}$$

Weighted graphs can be represented using adjacency-matrices or adjacency lists as well.

## Example

$G$:




Adjacency-Matrix:

$$\begin{bmatrix} 0 & 5 & 10 & 0 & 0 \\ 5 & 0 & 2 & 3 & 1 \\ 10 & 2 & 0 & 8 & 0 \\ 0 & 3 & 8 & 0 & 6 \\ 0 & 1 & 0 & 6 & 0 \end{bmatrix}$$

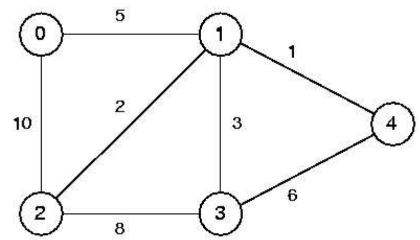
Use NIL instead of 0 if weights can be $< 0$



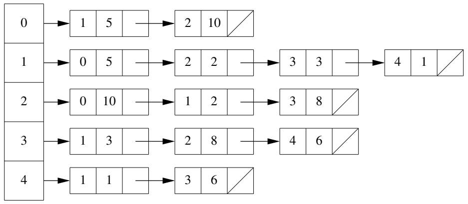


## Example

$G$:




Adjacency-List:







## References

**Graphs in Java**

- Java's standard libraries do not currently include implementations of graphs or graph algorithms

**Further Reading:**

- **Data Structures & Algorithms in Java**, Chapter 13
- **Introduction to Algorithms**, Chapter 23