

# Graph Theory

ian.mcloughlin@gmit.ie

Graphs

Paths and Cycles

Graph databases

Trees

# Graphs

# Seven Bridges of Königsberg



Is it possible to walk through the city crossing each of the seven bridges once and only once?

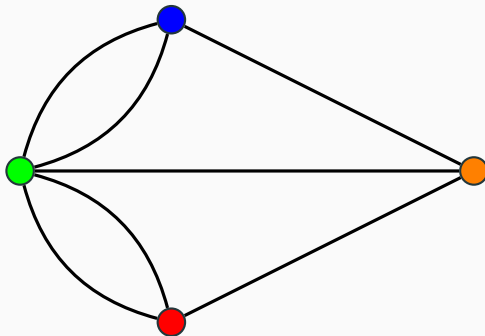
[www.nature.com/nbt/journal/v29/n11](http://www.nature.com/nbt/journal/v29/n11)



- Born 1707 in Basel, Switzerland.
- Euler's identity:  $e^{i\pi} + 1 = 0$ .
- Solved the Bridges of Königsberg problem.
- It's not possible to cross all bridges once and once only.

[https://en.wikipedia.org/wiki/Leonhard\\_Euler](https://en.wikipedia.org/wiki/Leonhard_Euler)

## Graph of Königsberg



## Definition

A *graph* consists of a finite set  $V$  and a set  $E$  of 2-subsets of  $V$ .

**Vertices** – the elements of the set  $V$  are called vertices.

**Edges** – the elements of  $E$  are called edges.

$G = (V, E)$  – this is the way we write the graph  $G$  consists of the vertex set  $V$  and the edge set  $E$ .

$$V = \{Green, Blue, Orange, Red\}$$

$$E = \{ \\ \{Green, Blue\}, \{Green, Blue\}, \{Green, Red\}, \\ \{Green, Red\}, \{Blue, Orange\}, \{Green, Orange\}, \\ \{Red, Orange\} \\ \}$$



## Adjacency list

Green	Blue	Orange	Red
Blue	Green	Blue	Green
Orange	Orange	Green	Orange
Red		Red	

# Defining different types of graphs

## Our definition of a graph

The definition given above for a graph is not consistent with looped edges, directed edges or repeated edges. We only need to make small changes to the definition of a graph to allow for directed edges and repeated edges.

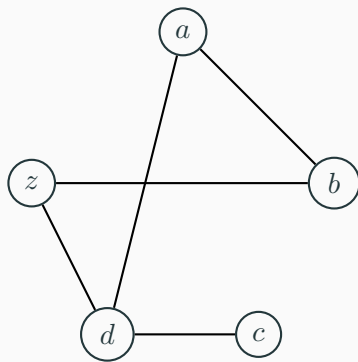
**Repeated edges** are edges that start and end at the same vertices.

**Directed edges** are edges where a direction is added.

**Looped edges** begin and end at the same vertex.

The application will determine the definition we want to use.

## A better example



### Exercise

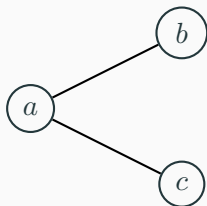
Determine the vertex set, edge set and adjacency list of this graph.

[global.oup.com/booksites/content/9780198507185/](http://global.oup.com/booksites/content/9780198507185/)

# Degree of a vertex

## Definition

The degree of a vertex is the number of edges that contain it.



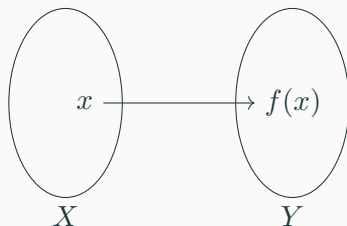
The degree of the vertex  $a$  is 2.

## Exercise

For each of the vertices on the previous slide, determine its degree.

## Definition

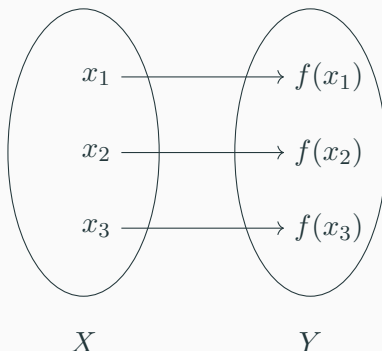
Suppose that  $X$  and  $Y$  are sets. We say we have a function  $f$  from  $X$  to  $Y$  if for each  $x$  in  $X$  we can specify a unique element in  $Y$ , which we denote by  $f(x)$ .



## Definition

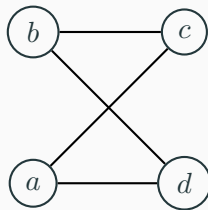
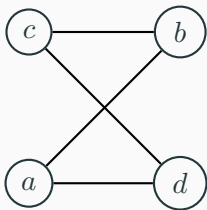
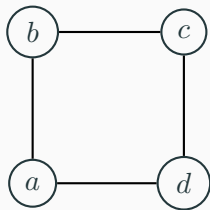
A bijection is function  $f$  from a set  $X$  to a set  $Y$  where both of the following are true:

- every  $y$  in  $Y$  is a value  $f(x)$  for at most one  $x$  in  $X$ .
- every  $y$  in  $Y$  is a value  $f(x)$  for at least one  $x$  in  $X$ .



## Definition

Two graphs  $G_1$  and  $G_2$  are said to be isomorphic when there is a bijection  $\alpha$  for the vertex set  $V_1$  of  $G_1$  to the vertex set  $V_2$  of  $G_2$  such that  $\{\alpha(x), \alpha(y)\}$  is an edge of  $G_2$  if and only if  $(x, y)$  is an edge of  $G_1$ .



# Sum of degrees

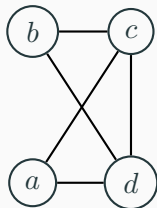
## Theorem

*The sum of the degrees of the vertices of a graph  $G = (V, E)$  is equal to twice the number of edges:*

$$\sum_{v \in V} \delta(v) = 2|E|$$

## Proof.

The degree  $\delta(v)$  of a vertex  $v$  is equal to the number of edges incident on it. Every edge is incident on two vertices. So every edge contributes 1 to the degrees of two distinct vertices. Therefore every edge contributes 2 to the sum total of the degrees of all the vertices.  $\square$

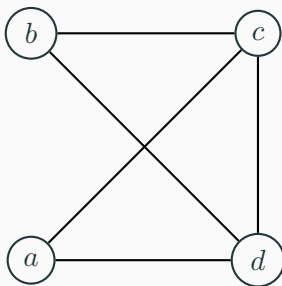




# Handshaking lemma

## Definition

A vertex is an odd vertex if its degree is odd, and it is an even vertex if its degree is even. The set of all odd vertices is denoted  $V_o$  and the set of all even vertices is denoted  $V_e$ .



## Exercise

Which of the above vertices are even, and which are odd?

## Lemma

*The number of odd vertices  $|V_o|$  in a graph is even.*

## Proof.

The sets  $V_o$  and  $V_e$  are disjoint (i.e. they don't have any elements in common.) Also, every vertex is either in  $V_o$  or  $V_e$ . Therefore  $V = V_o \cup V_e$  and  $|V| = |V_o| + |V_e|$ .

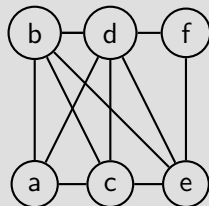
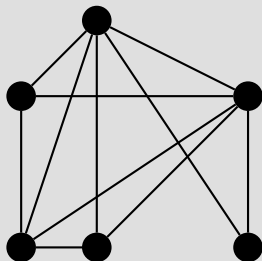
Furthermore:

$$\sum_{v \in V_o} \delta(v) + \sum_{v \in V_e} \delta(v) = 2|E|$$

Both  $2|E|$  and  $\sum_{v \in V_e} \delta(v)$  are even, so  $\sum_{v \in V_o} \delta(v)$  must be. Since  $\delta(v)$  is odd for every  $v$  in  $V_o$ , this must mean that  $|V_o|$  is even.  $\square$

## Exercise

Determine if these two graphs are isomorphic.



# Paths and Cycles

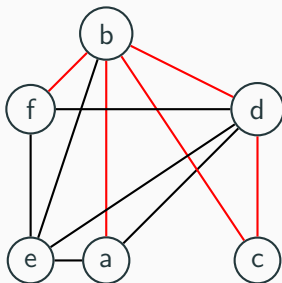
## Definition

A *walk* in a graph is a sequence of vertices

$$v_1, v_2, \dots, v_k$$

such that  $v_i$  and  $v_{i+1}$  are adjacent for  $1 \leq i < k$ .

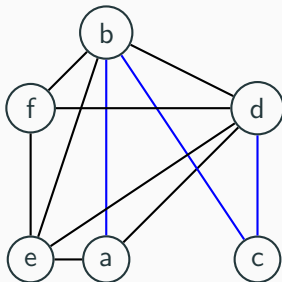
For instance, in the following graph  $a, b, c, d, b, f$  is a walk.



## Definition

A *path* is a walk where each vertex is distinct.

For instance, in the following graph  $a, b, c, d$  is a path.

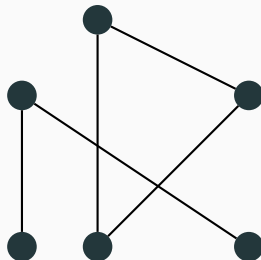
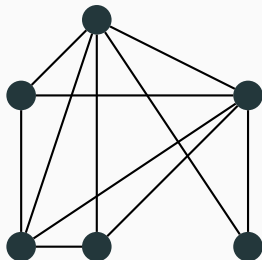


# Connected graphs

## Definition

A graph is *connected* if there is a path between each pair of vertices.

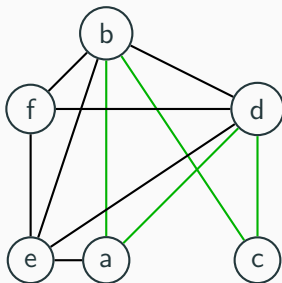
For instance, the graph on the left below is connected, but the one on the right is not.



## Definition

A *cycle* is a walk where each vertex is distinct, except for the start and end vertices being equal.

For instance, in the following graph  $a, b, c, d, a$  is a cycle.



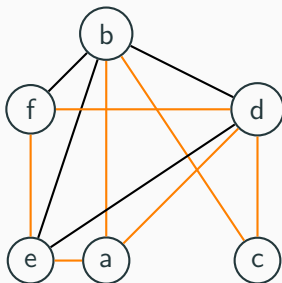


# Hamiltonian cycles

## Definition

A *Hamiltonian cycle* is a cycle that contains all of vertices of the graph.

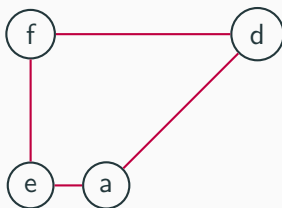
For instance, in the following graph  $a, b, c, d, f, e, a$  is a Hamiltonian cycle.



## Definition

An *Eulerian walk* is a walk which uses each edge of the graph exactly once.

For instance, in the following graph  $a, d, f, e, a$  is an Eulerian walk.



# Graph databases

## Definition

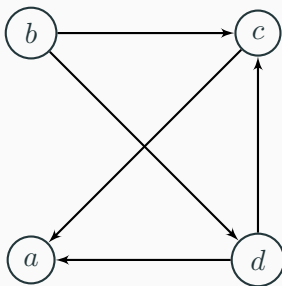
A *digraph* (short for directional graph) consists of a finite set  $V$  and a set  $E$  of ordered pairs of elements of  $V$ .

**Degrees** of vertices can now be split into in-degrees and out-degrees.

**Walks, paths, cycles** must be redefined.

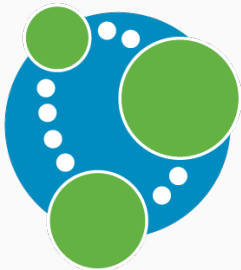
**Loops** are allowed in the above definition, unless we rule them out.

## Digraph example





- Neo4j is an open-source NoSQL graph database implemented in Java and Scala.
- Development started in 2003, it has been publicly available since 2007
- Available on GitHub.
- A graph is composed of two elements: a node and a relationship.



- Cypher is a declarative graph query language.
- What to retrieve from a graph, not on how to retrieve it.
- Allows for expressive and efficient querying and updating of the graph store.
- Cypher borrows its structure from SQL.

Create a node with the label User, and two properties:

---

```
1 CREATE (user:User { Id: 123, Name: "Jim" });
```

---

Find the node(s) with label User and their Id property being 123:

---

```
1 MATCH (user:User)
2 WHERE user.Id = 123
3 RETURN user;
```

---



Create a relationship with label FOLLOWS from user(s) with Id 123 to user(s) with Id 456:

---

```
1 MATCH (user1:User), (user2:User)
2 WHERE user1.Id = 123 AND user2.Id = 456
3 CREATE user1-[:FOLLOWS]->user2;
```

---

Create a relationship with label INVITED from user(s) with Id 123 to a new user with Id 789 and Name Jack:

---

```
1 MATCH (invitee:User)
2 WHERE invitee.Id = 123
3 CREATE invitee-[:INVITED]->(invited:User {Id: 789,
4                                     Name: "Jack"});
```

---

Delete all nodes:

---

```
1 MATCH (x)
2 DELETE x;
```

---

- Suppose we have nodes representing people.
- We give them the label People.
- We also want to identify each person as either Male or Female.
- Should we use Male and Female labels, or a Gender property?
- If you are going to use the person's gender in a lot of queries, a normal property will be relatively slow, so you should use a label.
- However, you can also index some of your properties to high-light them as important.

Find the minimum number of hops between two nodes.

---

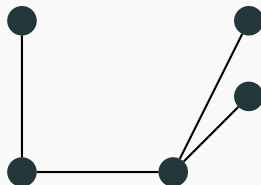
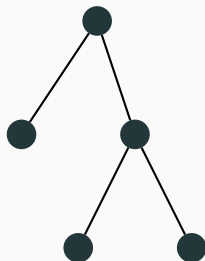
```
1 MATCH p=shortestPath(  
2   (a:Actor {id: 1})-[*]-(b:Actor {id: 10})  
3   )  
4 RETURN p;
```

---

# Trees

## Tree

A *tree* is a graph where every pair of vertices has a path between them, and there are no cycles.



# Levels

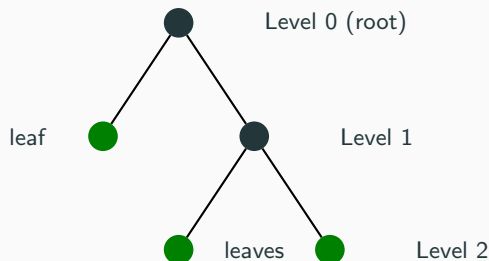
**Root** Specified vertex for some purpose.

**Levels** Root is at level 0, neighbours of the root are at level 1, their other neighbours at level 2, and so on.

**Height**  $h$ , where vertex at level  $h$  but not at level  $h + 1$ .

**Leaf** Vertex at level  $i$  with no neighbours at level  $i + 1$ .

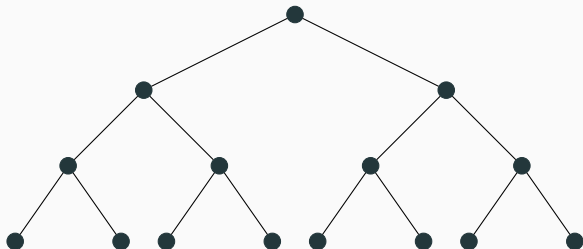
**Internal vertex** A non-leaf.





## Definition

When a vertex at level  $i$  is connected to a vertex at level  $i + 1$  it's common to call the former the *father* and the latter the *son*. A rooted tree is  $m$ -ary if every father has the same number of sons. A 2-ary rooted tree is called a *binary tree*.



## Definition

Two rooted trees are said to be *isomorphic* if there is an isomorphism between them considered which takes the root of one tree to the root of the other.

# Height of an $m$ -ary rooted tree

## Theorem

*The height  $h$  of an  $m$ -ary rooted tree with  $l$  leaves is at least  $\log_m l$ . That is:*

$$h \geq \log_m l$$

## Proof.

First note that:  $h \geq \log_m l \Leftrightarrow m^h \geq m^{\log_m l} \Leftrightarrow m^h \geq l$ .

So, we just need to show the number of leaves is at most  $m^h$ . For a tree of depth 0, there is only one vertex and  $m^0 \leq 1$ . Suppose we know that the theorem is true for trees of height  $h - 1$ .

Consider a tree of height  $h$ , with  $l$  leaves. We can create  $m$  trees of height  $h - 1$  from it by deleting the root. Each of these smaller trees has at most  $m^{h-1}$  leaves, and there are  $m$  of them. So the big tree has at most  $m \times m^{h-1} = m^h$  leaves.



## Definition

A decision tree is a rooted tree where each vertex represents a decision. The possible results of each decision are represented by the edges connecting the vertex to the vertices at the next level down. Final outcomes of the procedure are represented by the leaves.

