# MATH1510 - Discrete Mathematics
# Graphs

University of Newcastle



## Definitions

In a graph $G$:

**Path** Specified by a list that alternates vertices and edges: $v_0, e_1, v_1, e_2, v_2, \ldots, e_n, v_n$, where edge $e_1$ is incident on vertices $v_0$ and $v_1$, edge $e_2$ is incident on vertices $v_2$ and $v_3$, and so on

**Initial vertex** Is where the path starts: $v_0$

**Terminal vertex** Is where the path end: $v_n$

**Length** The number of edges in the path. [Length $= 0$?]

**Cycle** A path with length $n > 0$, $v_0 = v_n$ and no edge repeated.

If the graph is simple, a path can be specified purely by its *list of vertices* $v_0, \ldots, v_n$, as the edges can be inferred.



## Existence of cycles

**Theorem**

*A graph $G$ with finitely many vertices each of degree $\geqslant 2$ contains a cycle.*

**Proof.**

Start at any vertex $v_0$ and find an edge $e_1$ incident on it. If the edge is a loop, $v_0, e_1, v_0$ is a cycle. Otherwise, $e_1$ is incident on another vertex $v_1$. Add both to the path. Since $\delta(v_1) \geqslant 2$, another edge $e_2$ is incident on it, with another vertex $v_2$. Add both to the path. Continue in this manner. Since there are only finitely many vertices, eventually some $v_s$ will be equal to a previous $v_r$. Then $v_r, e_{r+1}, v_{r+1}, \ldots, v_s = v_r$ will be a cycle. □

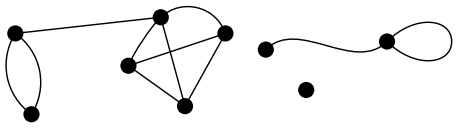
**Note**

Not every vertex has to be contained in a cycle.



## Existence of paths: Connectivity and distance

Two vertices in a graph are connectable if there is at least one path between them. Given two connectable vertices, we defined the distance to be the length of the shortest path between them.
A graph $G$ is connected if all pairs of vertices are connectable. Otherwise $G$ is disconnected and is made up of many connected parts called connected components.



## Simple paths

A path or cycle is simple if it has no vertex repeated (except possibly the initial and terminal vertices of a cycle). The cycle obtained in the proof of the cycle existence theorem is a simple cycle.

Indeed, the existence of simple paths wherever there are paths is guaranteed by the following theorem.



**Theorem**

*If $u$ and $v$ are vertices of a graph $G$ and there is a path (or cycle) from $u$ to $v$ of length $n$ then there is a simple path (or cycle) from $u$ to $v$ whose length is $\leqslant n$.*

**Proof.**

Cycles first. Of all the cycles from $u$ to $u$, choose one of minimum length, say $v_0, e_1, v_1, \ldots, e_m, v_m$ where $v_0 = v_m = u$. If $v_i = v_j$ and $i < j$, then $v_0, e_1, \ldots, e_i, v_j, e_{j+1}, \ldots, e_m, v_m$ is a cycle of length $i + m - j < m$, contradicting the initial choice of a shortest cycle. Hence all the $v_i$'s are distinct and the cycle is simple. For a path with $u \neq v$, add a vertex $w$ and two edges $(w, u)$ and $(w, v)$ to form a graph $G'$. Then there is a cycle from $w$ to $w$ hence a simple cycle from $w$ to $w$ and hence a simple path from $u$ to $v$. □



## Two interesting features of the proof

Extremal principle. Among finitely many elements we can always choose one with some extremal property (in this case: a shortest cycle).

Reduction to previous case. If we want to prove two (or more) similar statements, sometimes the second statement can be reduced to the first statement by a slight modification. In this case: The existence of a simple *path* in the original graph follows immediately from the existence of a simple *cycle* in a modified graph.



## Definitions

We revisit and complete Euler's Königsberg Bridge problem. Given a graph $G$:

Euler path A path in $G$ containing every edge of $G$ exactly once;

Euler cycle An Euler path that is a cycle;

Eulerian graph A graph which has an Euler cycle.

From the discussion in week 1, every vertex in an Eulerian graph has even degree. This is almost the whole story.

## Euler's theorem

**Theorem**

*A graph has an Euler cycle if, and only if,*

1. *every vertex has even degree, and*
2. *all the vertices with nonzero degree are in the same connected component.*

**Proof of "⇒".**

From the discussion in week 1, we know that in an Eulerian graph, the degree of every vertex is even. Clearly, an Eulerian graph also has to satisfy the second condition. □

- This was the easy part.
- For the converse it is sufficient to prove that every connected graph in which every vertex has even degree has an Euler cycle.



## Reminder: Strong induction

**Set-up** We set up $P(n)$ to be a statement we want to prove for all $n$.

**Base step** Prove the statement for the first value of $n$ – usually $P(1)$.

**Inductive step** Assume the statement is true for all integers up to $k$, i.e., assume $P(1)$, $P(2)$, ..., $P(k-1)$, $P(k)$ are all true. Use this assumption to prove that $P(k+1)$ is true.

**Conclusion** $P(n)$ is true for all $n$.



## Proof "⇐" (strong induction on the number of edges)

**Set-up** $P(n)$ : "Every connected graph $G$ with $n$ edges and with every vertex of even degree has an Euler cycle."

**Base step** For $n = 1$, the graph is a single loop, which has an Euler cycle.

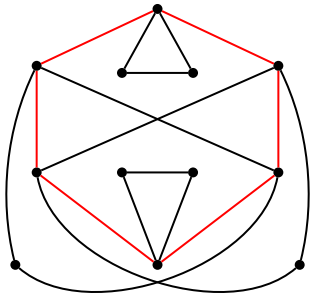
**Inductive step**

- Assume $P(j)$ is true for all $j \leqslant k$ and let $G$ be a connected graph with $k+1$ edges and with every vertex of even degree.
- Every vertex has degree $\geqslant 2$, hence there exists a cycle $C$ in $G$.
- Let $G'$ be the graph $G$ with the edges from $C$ removed, and let $G_1, \ldots, G_m$ be the connected components of $G'$.
- Each of $G_1, \ldots, G_m$ has $\leqslant k$ edges, is connected, and has every vertex of even degree. So by the inductive hypothesis, each $G_j$ has an Euler cycle $C_j$. By grafting each cycle $C_j$ onto the main cycle $C$, we create an Euler cycle for $G$.



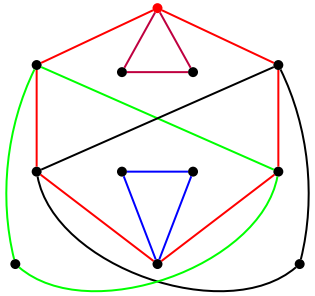
## How many connected components . . .

after deleting the red cycle?



A 1
B 2
C 3
D 4

Starting at the red vertex and choosing the red cycle as the base cycle, which order of traversing the components gives an Euler cycle?



A black – green – blue – purple

B black – blue – green – purple

C black – green – purple – blue

D black – purple – blue – green



## Proof "⇐" (continued)

### Conclusion

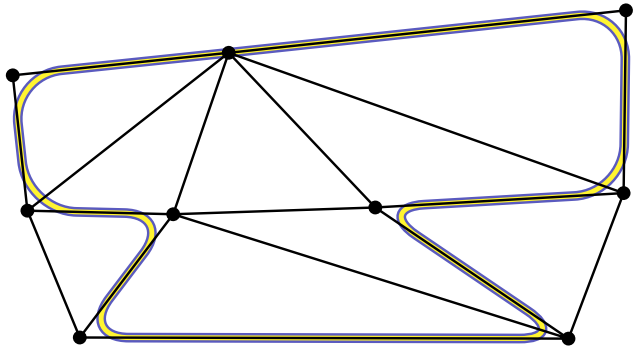
$P(n)$ is true for all $n$, i.e., every connected graph $G$ with every vertex of even degree has an Euler cycle. □

### Corollary

*A graph has an Euler path between two vertices $u \neq v$ if, and only if,*

1. *it is connected, and*
2. *$u$ and $v$ have odd degree and all vertices in $V - \{u, v\}$ have even degree.*





A simple cycle in a graph $G$ which contains every vertex of $G$ is called a Hamiltonian cycle. If a graph has such a cycle, it is called a Hamiltonian graph.

**Q** How can we tell if a graph has a Hamiltonian cycle?

**A** It's hard to say.



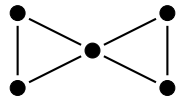
## Necessary conditions for Hamiltonian circuits

If a graph $G$ has a Hamiltonian cycle, then it:

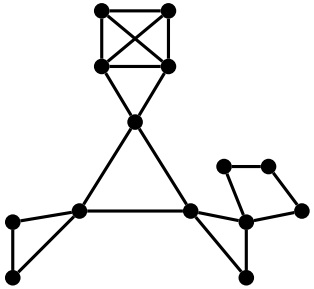
- Must be connected;
- Can't have any vertices of degree 1;
- Can't have any *articulation points*.

### Definition

An articulation point is a vertex whose deletion increases the number of connected components.



## How many articulation points?



A 2
B 3
C 4
D 5



## A sufficient condition

The problem of finding Hamiltonian circuits is *hard*:

- the commonsense exhaustive-search algorithm requires a lot of work for large graphs
- more intelligent algorithms exist, but they are very complex, and only marginally better than exhaustive search
- there is no known mathematical criterion that is both necessary and sufficient

We have seen some conditions that are necessary but not sufficient for a graph to be Hamiltonian (connected, no articulation points, etc.)



## A sufficient condition

One example of a condition that is sufficient (but not necessary) is Dirac's Theorem.

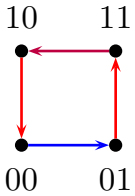
**Theorem (Dirac's Theorem)**

*Let $G$ be a simple graph with $n \geqslant 3$ vertices. If every vertex has degree $\geqslant n/2$, then $G$ has a Hamiltonian cycle.*

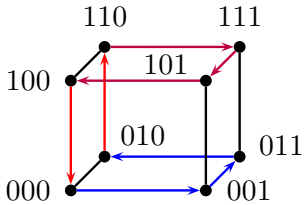


## Gray codes

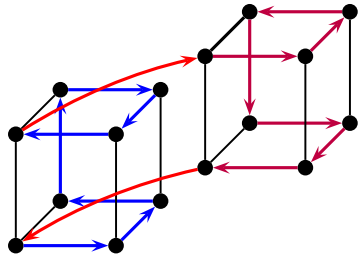
The $n$-cube has a Hamiltonian cycle called Gray Code. This leads to a counting pattern for the numbers $0, 1, \ldots, 2^n - 1$.




$Q_2$: 0, 1, 3, 2




$Q_3$: 0, 1, 3, 2, 6, 7, 5, 4



$Q_4$: 0, 1, 3, 2, 6, 7, 5, 4, 12, 13, 15, 14, 10, 11, 9, 8

Despite superficial similarities, the problems of finding Euler cycles and Hamiltonian cycles are substantially different — the latter is far more difficult.
For a general graph with $n$ vertices,

- An algorithm such as in the proof of the characterization of Eulerian graphs gives an Euler cycle within a time that is a *linear* function of $n$.
- The best known algorithm for finding a Hamiltonian cycle uses a number of steps which is an *exponential* function of $n$.

Some examples of different rates of growth are given below.
In general, questions of *complexity* of an algorithm or a problem are important. What we are interested in is how the amount of computation required grows *as a function of the size of the problem*.



## Examples of different growth rates

Some sample rates of polynomial and exponential growth (values are approximate)

| $n$ | $100n$ | $n^2$ | $n^3$ | $n^{10}$ | $2^n$ | $1.1^n$ |
|---|---|---|---|---|---|---|
| 10 | 1000 | 100 | 1000 | $10^{10}$ | $10^3$ | 2.6 |
| 20 | 2000 | 400 | 8000 | $10^{13}$ | $10^6$ | 6.7 |
| 30 | 3000 | 900 | 27000 | $10^{15}$ | $10^9$ | 17 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 100 | $10^4$ | $10^4$ | $10^6$ | $10^{20}$ | $10^{30}$ | $10^4$ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 1000 | $10^5$ | $10^6$ | $10^9$ | $10^{30}$ | $10^{301}$ | $10^{41}$ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 10000 | $10^6$ | $10^8$ | $10^{12}$ | $10^{40}$ | $10^{3010}$ | $10^{414}$ |
| | | | polynomial | | exponential | |



Some other (faster and slower) rates of growth

| $n$ | $n!$ | $n^n$ | $\log_2 n$ | $\log_{10} n$ | $n \log_2 n$ |
|---|---|---|---|---|---|
| 10 | $10^7$ | $10^{10}$ | 3.3 | 1 | 33.2 |
| 20 | $10^{18}$ | $10^{26}$ | 4.3 | 1.3 | 86.4 |
| 30 | $10^{32}$ | $10^{44}$ | 4.9 | 1.5 | 147 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 100 | $10^{158}$ | $10^{200}$ | 6.6 | 2 | 664.4 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 1000 | $10^{2568}$ | $10^{3000}$ | 10 | 3 | $10^4$ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 10000 | $10^{35659}$ | $10^{40000}$ | 13 | 4 | $10^5$ |
| | | | logarithmic | | |

$\log \log \log n$ has been proved to tend to infinity with $n$, but has never been observed to do so.



## Sorting

A simpler example of complexity is that of Sorting Algorithms, i.e., algorithms that solve the problem to arrange $n$ items in some systematic way, for instance to sort a list of words alphabetically.

- "Commonsense" algorithms have complexity of order $n^2$.
- Better algorithms have complexity of order $n \log_2(n)$.

## Travelling Salesman Problem

If $G$ is a *weighted* graph, the Travelling Salesman Problem is to find a Hamiltonian cycle in $G$ having the *minimum weight*. Of all the possible Hamiltonian cycles in the graph, find one in which the sum of the weights on all edges in the cycle is the smallest.
The name (aka TSP) comes from the concept of a salesman wanting to find the shortest route to visit every city and return home.



## Example

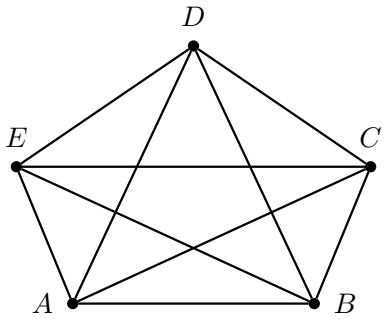
A production line in a factory produces 5 types of component, and needs a rotating monthly schedule that devotes one block of time to each product. The cost of reconfiguring is given in the following table. Design a minimum-cost schedule.

| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 2 | 5 | 12 | 6 |
| B | 2 | 0 | 3 | 11 | 4 |
| C | 5 | 3 | 0 | 9 | 6 |
| D | 12 | 11 | 9 | 0 | 7 |
| E | 6 | 4 | 6 | 7 | 0 |

Here it is possible to use *Exhaustive search* find all Hamiltonian cycles, and add up each.
This algorithm has complexity $n!$. Although there are more efficient algorithms for TSP, all known algorithms have *at least exponential* complexity.



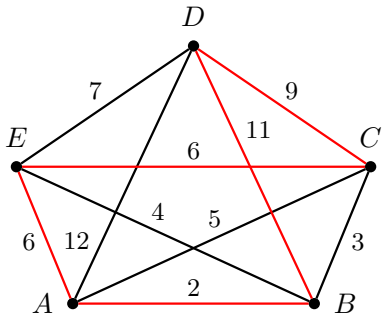
## How many Hamiltonian cycles in $K_5$?




A 5
B 12
C 24
D 120



## What is the weight of the cycle $(A, B, D, C, E, A)$?




A 30
B 34
C 38
D 42

## Summary

**Euler paths and cycles.** Paths (resp. cycles) covering every edge exactly once

**Hamilton paths and cycles.** Paths (resp. cycles) visiting every vertex exactly once

**Gray Code** A special Hamilton path in the $n$-cube

**Complexity.** Classification of computational problems according to how the solve time grows with instance size