# CO150 - Graphs and Algorithms

## Prelude

The content discussed here is part of CO150 - Graphs and Algorithms (Computing MEng); taught by Iain Phillips, in Imperial College London during the academic year 2018/19. The notes are written for my personal use, and have no guarantee of being correct (although I hope it is, for my own sake). This should be used in conjunction with the notes.

## 14th January 2019

Introduction to the structure of the course;

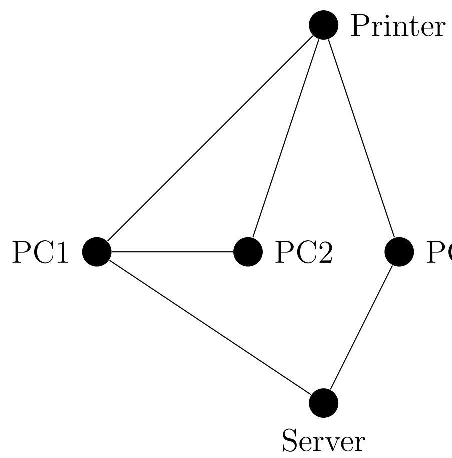
Part I: Graphs

Part II: Graph Algorithms

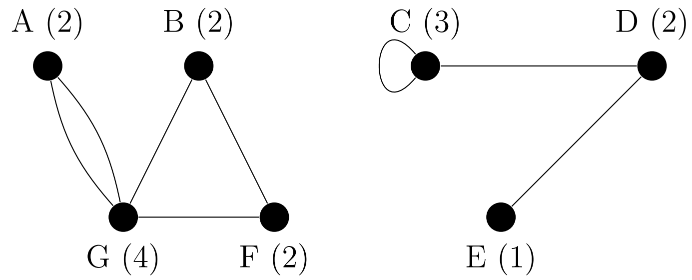
Part III: Algorithm Analysis

Part IV: Introduction to Complexity

An example graph with a real life application;




Note how all the PCs are directly connected to the printer, but PC2 can only reach the server through PC1. On the other hand, we can create a more general graph to display some features that may be less common;




Note that this isn't actually two graphs; it's **disconnected components**. Between A, and G, there are two **parallel arcs / edges**, and C has **loop** with itself. I will continue to refer to this graph as the "example", for the remainder of this section, since it displays properties which we may want to analyse later.

We can say that the left subgraph is robust, as it will remain connected against a single failure. However, the right subgraph isn't robust, as a failure between C, and D, or between E, and D would cause one of the nodes to become disconnected. We can then remedy this by adding a connection between C, and E.

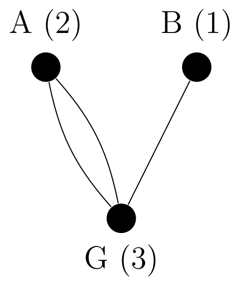
In the graph drawn above, the degrees are also specified - which is the number of arcs connected to it. Note that the degree of C is 3, as we count loops twice for consistency reasons. The sum of the degrees is 16, which is double the number of arcs (8). This is because each arc is counted twice (where it starts, and where it ends), therefore the sum of the degrees is always even. From that, we can then infer that the number of odd nodes (C, and E in our case) must be even. This is trivial to prove with arithmetic.

### Subgraphs

We can say that $G_1$ is a subgraph of $G_2$ if both of the following criteria apply;

- $\text{nodes}(G_1) \subseteq \text{nodes}(G_2)$
- $\text{arcs}(G_1) \subseteq \text{arcs}(G_2)$

A full (induced) subgraph occurs when we have a set of nodes, $X$, such that $X \subseteq \text{nodes}(G)$. Every connection between the nodes in $X$, that was present in $G$, exists in $G[X]$. Then $G'$ is a full subgraph of $G$, if $G' = G[X]$ for some $X$. For example, let $X = \{A, B, G\}$, from the example graph, then we have the following induced subgraph;




If we have some subgraph $G'$, and $\text{nodes}(G') = \text{nodes}(G)$, then it $G'$ spans $G$.

### Adjacency Matrix

For the entry in the matrix $a_{i,j}$, it represents the number of arcs thast connect $i$ to $j$. In an undirected graph, this matrix is symmetric (such that $a^\top = a$). In our example, we're doing the rows, and columns, alphabetically. It's also important to note that we count each loop twice in a diagonal entry. We can determine the degree of a node by taking the sum of its respective row (or column), and find the number of arcs by taking the sum of all the values in the matrix, and then halving it.

$$\begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \\ G \end{array} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 2 & 1 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

### Adjacency Lists

You'll notice that in our graph, we have a lot of 0s, which makes it less efficient to store as an adjacency matrix; especially if we don't require random access to the degrees. We tend to use $n$ to represent the number of nodes (vertices), and $m$ to represent the number of arcs (edges). You'll note that the size of this is $\leq n + 2m$ (as we have $n$ nodes on the left, and each arc is counted twice, except for loops). Therefore, we can say a graph is sparse if $2m \ll n^2$. Since certain algorithms we work with only look at the arcs incident to a given node, a linked list will be better for sparse graphs.

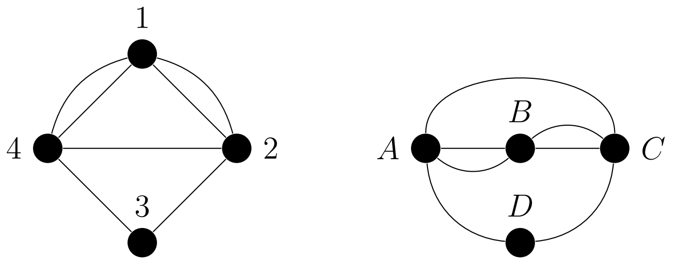
| | |
|---|---|
| A | $\to$ G, G |
| B | $\to$ F, G |
| C | $\to$ C, D |
| D | $\to$ C, E |
| E | $\to$ D |
| F | $\to$ B, G |
| G | $\to$ A, A, B, F |

### Big-Oh Notation

I'm too lazy to write out the example, but the idea is that we ignore constant factors, and only consider the most significant term; for example, we could summarise some algorithm that takes $3n^4 + 2n - 4631$ to run as $O(n^4)$. This has significant advantages, since it allows us to abstract away from the implementation / hardware specifics, and instead focus on the factors which determine growth.

### Isomorphism

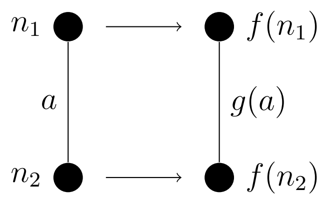
In general, an isomorphism is a bijection that preserves connections. While the two graphs drawn below appear fairly different, they are isomorphic. Mapping from the left, to the right, we know that $3 \mapsto D$, simply because they are the only nodes with degree 2. It's also evident that $1 \mapsto B$, as it's the only node which has two sets of parallel arcs coming out of it. However, it doesn't matter which of 4, or 2, maps to $A$, or $C$. Therefore, we can say $4 \mapsto A, 2 \mapsto C$, or $4 \mapsto C, 2 \mapsto A$.




While we're able to check this fairly easily by simply looking at the graph, a computer would have to rearrange the LHS' adjacency matrix to the RHS' (or vice versa).

Given two graphs, $G, G'$, an isomorphism from $G$ to $G'$ consists of two bijections (one-to-one mapping), as well as an additional restriction;

- $f$ : nodes$(G) \mapsto$ nodes$(G')$
- $g$ : arcs$(G) \mapsto$ arcs$(G')$
- if $a \in$ arcs$(G)$, with endpoints $n_1, n_2$, then the endpoints of $g(a)$ are $f(n_1), f(n_2)$ (see the diagram below for a visual example).




In order to confirm whether two graphs are isomorphic, the easiest approach is to first check the obvious; whether the number of arcs, nodes, and loops are the same, as well as the degrees of the nodes. If any of these are different, then the graphs cannot be isomorphic. However if they pass all the tests, then we can attempt to find a bijection on the nodes.

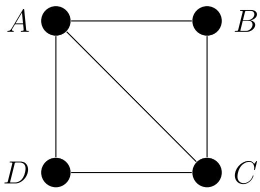
## 17th January 2019

### Complexity

Generally, the process of determining whether two graphs are isomorphic is computationally expensive, hence it has a high complexity. A naive approach would be to check all the permutations, which would then lead to a time complexity of $O(n!)$, which is worse than even exponential ($O(2^n)$).

### Automorphisms

An automorphism on $G$ is an isomorphism from $G$ to itself. Every graph has at least one automorphism (the identity). Consider the following graph;

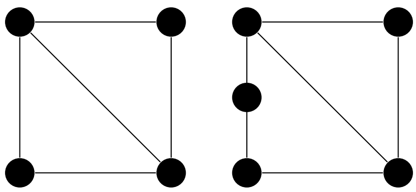



We can do the following method to find the number of automorphisms;

- fix a node, $B$, it can go to where $D$ is, or stay (2 possibilities)
- take the next node $A$, it can either stay where it is, or go to where $C$ is (2 possibilities), now fix it
- take the next node $C$, it can only stay where it is, as it can't go to $D$ since $D$ isn't connected to $B$, nor does it have a degree of 3 (1 possibility), now fix it
- finally $D$ can only stay where it is (1 possibility)
- multiply all the possibilities, and we have 4 automorphisms

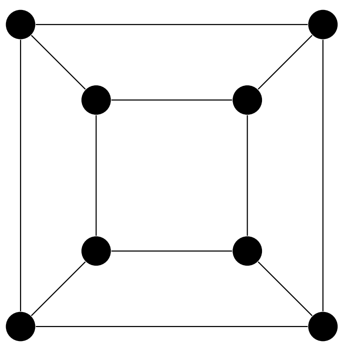
### Planar Graphs

We can say a graph is planar if it can be drawn such that no arcs cross. Any non-planar graph contains $K_5$, or $K_{3,3}$ as a subgraph homeomorphic. We can say that two graphs are **homeomorphic** if they can be obtained by a series of operations such that an arc $x - y$, is replaced by two arcs $x - z$, and $z - y$. For example, the two graphs below are homeomorphic.



There is a linear time algorithm to check whether a graph is planar; however in this case linear time means $O(n + m)$, with the previous definitions.

Any planar graph splits the plane into regions, which are referred to as faces; the graph below splits it into 6 faces (including the outside region). With a graph $G$ that has $N$ nodes, $A$ arcs, and $F$ faces, Euler's formula states $F = A - N + 2$ for any connected planar graph.



### Graph Colouring

Any (literal, real-life) map can be convereted into a simple planar graph by letting the countries represent nodes, and joining them if they are neighbours. This newly generated graph is known as the dual graph. We can say some graph $G$ is $k$-colourable, if the nodes of $G$ can be coloured with no more than $k$ colours, therefore every simple planar graph is 4-colourable.

### Bipartite Graphs

We can say a graph is bipartite if we can partition nodes($G$), into two sets $X$, and $Y$, such that no two nodes of $X$ are joined, and likewise for $Y$. A graph is biparite $\Leftrightarrow$ it is 2-colourable.

### Paths, and Connectedness

A path in a graph is a sequence of adjacent arcs, although normally described by the nodes that we pass through. A path is called **simple** if it doesn't repeat nodes, and a graph is **connected** if there is a path joining any two nodes.

We can define a relation on nodes($G$) by $x \sim y \Leftrightarrow$ there is a path from $x$ to $y$. This is an equivalence relation, as we can prove it's reflexive, symmetric, and transitive.

- $\forall x \in \text{nodes}(G)[x \sim x]$, $x$ is trivially connected to itself, hence it is reflexive
- $\forall x, y \in \text{nodes}(G)[x \sim y \Rightarrow y \sim x]$, as we are working on an undirected graph, this follows trivially
- $\forall x, y, z \in \text{nodes}(G)[x \sim y \land y \sim z \rightarrow x \sim z]$, follows trivially by definition of paths

A cycle (circuit) is a special type of path that finishes where it starts, has at least one arc, and doesn't reuse an arc. A graph which doesn't have cycles is **acyclic**.

## 21st January 2019

### Euler Paths / Circuits

An Euler path is a special type of path where each arc is used exactly once, and an Euler circuit is a cycle which uses each arc exactly once (therefore an EC is an EP which finishes at the start node). A connected graph has an EP $\Leftrightarrow$ there are 0, or 2 odd nodes, and there is an EC $\Leftrightarrow$ there are no odd nodes.