D1 Graphs Glossary!

How to use this glossary:

LOOK (Read through the examples, and make sure you understand the information, and the examples)

COVER the whole thing (try to remember all columns, including the number of objects, and their names)

WRITE out all the columns on a blank sheet of paper (The examples should not be the same every time!)

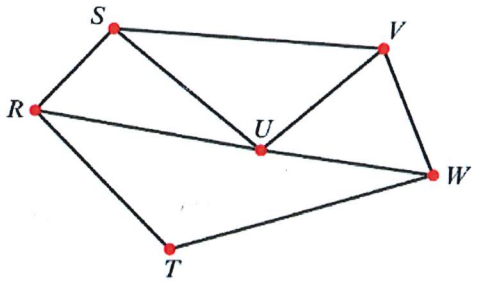
CHECK your work (make sure you are LOOKing properly and that you've captured every part of the definition)

[1] Getting around Graphs

Object	Definition
Walk	A sequence of edges, in which the end of each edge is the beginning of the next (except the last edge).
Trail	A walk in which no edge is repeated.
Path	A trail in which no vertex is repeated.
Cycle	A path which is a closed (the final vertex is also the start vertex).
Hamiltonian Cycle	A cycle which visits every vertex. (Since it is also a path, each vertex is visited precisely once).

In the graph below, an example of:

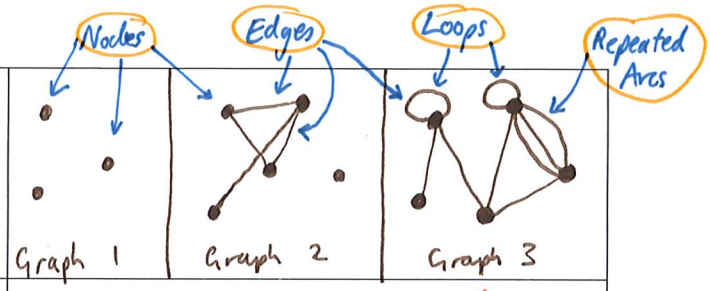
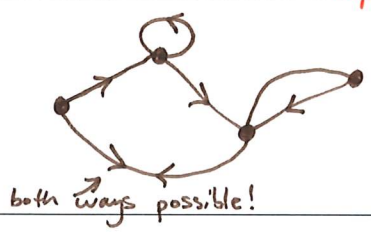
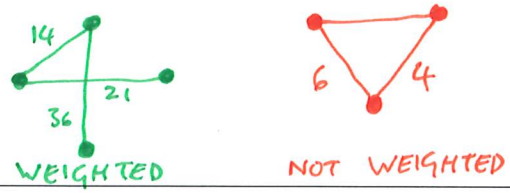
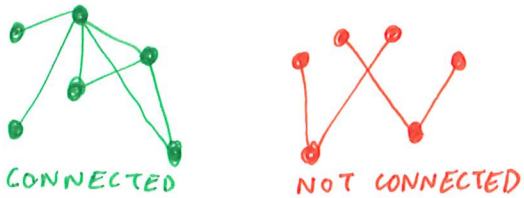
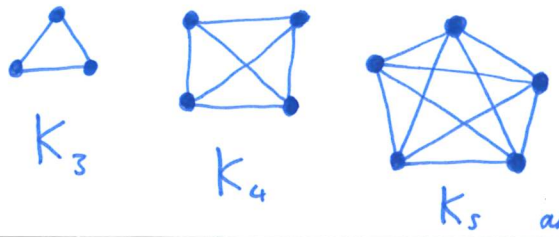
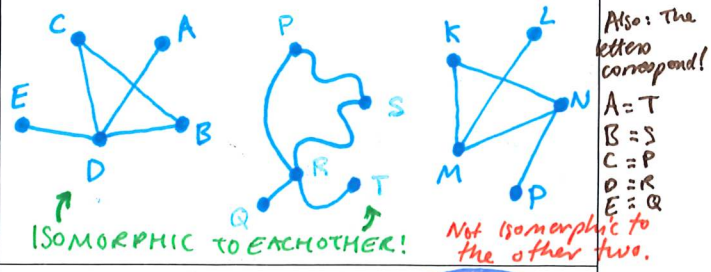
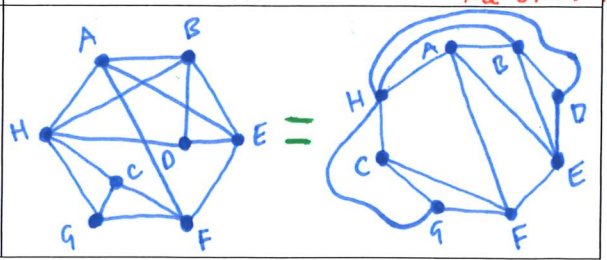
- a **walk** is $RSUWVU$ — It is ok to include a vertex or an edge more than once on a walk.
- a **path** is $RSUVW$
- a **trail** is $RUSVUW$
- a **cycle** is $RSUR$ — It is not necessary to include every vertex in a cycle.
- a **Hamiltonian cycle** is $RSUVWTR$.



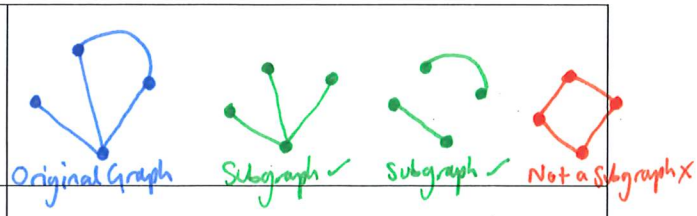
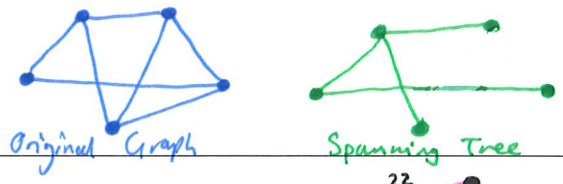
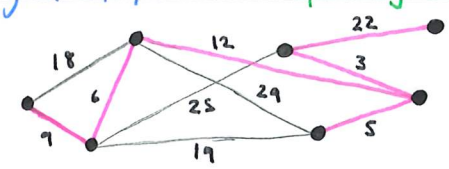
* Note that all Hamiltonian Cycles are Cycles, all Cycles are Paths, all Paths are Trails, and all Trails are Walks. (So all Cycles are Walks etc.)

[2] Adjectives to Describe Graphs

- Order (of a vertex) is the number of arcs which connect to it. (also known as degree or valency)

Graph	A graph consists of a discrete number of **vertices/nodes**, which are connected by a discrete number of **edges/arcs**. *This example also illustrates a **loop**.	
Simple Graph	A graph in which there are no loops and in which there is no more than one edge connecting any pair of vertices.	Simple ✓ Simple ✓ Not Simple ✗ (loops and multiple repeats of the same arc)
Directed Graph NB: There is such a thing as the in-degree and out-degree, but its not on your spec.	A graph in which at least one edge has a direction associated with it.	
Weighted Graph	A graph in which all edges have a weighting associated with them.	
Connected Graph	A graph is connected if a path exists between every pair of vertices	
Complete Graph	A complete graph is a simple graph in which every pair of vertices is connected by an edge.	
Isomorphic Graphs	Two graphs are isomorphic if one can be stretched, twisted or otherwise distorted into the other. In the diagram (right), graphs 1 and 2 are isomorphic to one another, but graph 3 is not isomorphic to them.	
Planar Graph	A graph which can be drawn without any edges crossing.	

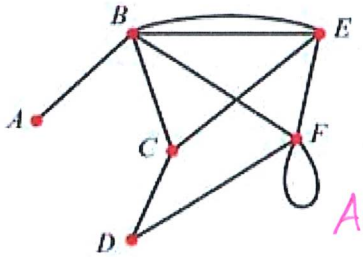
[3] Further Types of Graph

Subgraph	If you delete vertices/edges from a graph (but don't add any), the resulting object is a subgraph of the original graph.	
Tree	A tree is a simple connected graph with no cycles.	
Spanning Tree	A subgraph which includes all the vertices of the original graph (spans it), and is also a tree.	
Minimum Spanning Tree (MST)	A spanning tree of a weighted graph, such that the total length of its arcs is as small as possible.	

[4] Other Objects

Adjacency Matrix (Incidence Matrix)	Each entry in an adjacency matrix describes the number of arcs joining the corresponding vertices.
Distance Matrix	The entries represent the weight of each arc, not the number of arcs.
Euler's Handshaking Lemma	For any undirected graph: $\sum_{v \in V} \deg(v) = 2\lvert E \rvert$

* V = set of all vertices
* E = set of all edges

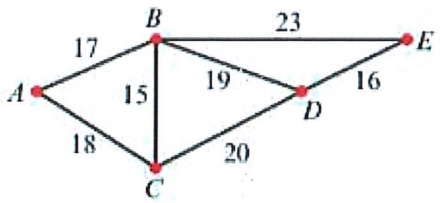


Adjacency Matrix

	A	B	C	D	E	F
A	0	1	0	0	0	0
B	1	0	1	0	2	1
C	0	1	0	1	1	0
D	0	0	1	0	0	1
E	0	2	1	0	0	1
F	0	1	0	1	1	2

This 2 indicates 2 direct connections (arcs) between B and E.

Because this 2 is on the leading diagonal, it indicates the presence of a loop.



Distance Matrix

	A	B	C	D	E
A	—	17	18	—	—
B	17	—	15	19	23
C	18	15	—	20	—
D	—	19	20	—	16
E	—	23	—	16	—

* If the matrix is not symmetrical, then the network is directed (and vis a versa).

Chapter 4 – The Route Inspection Algorithm

For a connected graph (all disconnected graphs are non-Eulerian):

The Graph is Eulerian ⟺ It contains an Eulerian Circuit (a trail which ① includes every edge, and ② starts/finishes at the same vertex). ⟺ All vertices have even degree

Eulerian circuits are not truly cycles.

it must necessarily touch every vertex if it touches every edge.

The solution to the Route Inspection algorithm in this case will be precisely the Eulerian cycle. Therefore the length of the shortest route will be equal to the total weight of the network.

The Graph is Semi-Eulerian ⟺ It contains a trail which ① includes every edge, but ② starts/finishes at different vertices. ⟺ Precisely 2 vertices have an odd degree

In this case, the length of the shortest route that the route inspection algorithm finds will be equal to the total weight of the network + the shortest route between the 2 odd vertices.

The circuit ~~cannot~~ must start on one of the odd vertices and end on the other, because of "belly button logic" (inny-outy-inny or outy-inny-outy).

The Graph is "non-Eulerian" (neither Eulerian or semi-eulerian). ⟺ It has 1, 3, 4, 5, 6, ... odd vertices

The Route Inspection Algorithm (or Chinese Postman algorithm), can be used to find the shortest route in a network which traverses ① every arc at least once, and ② returns to its starting vertex.

↳ If the network has more than 4 odd nodes, additional information will be provided that will restrict the number of pairings that you need to consider!

Chapter 5 – The Travelling Salesman Problem

If upper bound = lower bound, then you have an optimal solution!

In the Classical Problem each vertex must be visited exactly once before returning.

In the Practical Problem each vertex must be visited at least once before returning to the start.

all classical solutions are also practical solutions

only for undirected networks

Create a table of Least Distances (S.1)

You want to find a LOWER BOUND (S.3)

for the classical problem

'Delete' a designated arc (delete the row & column)

Find an RMST using Prim or Kruskal (called residual because it's the MST that's left when you delete the vertex).

Connect the deleted vertex to the RMST using the 2 shortest possible arcs ~~to create an MST.~~ ← not true. A tree has no cycles.

If you do not have a Hamiltonian cycle, you have not got a solution. The weight of the ~~MST~~ graph forms a lower bound.

You have an Hamiltonian cycle. Your ~~MST~~ graph is an optimal solution.

You want to find an UPPER BOUND

for the practical problem

don't confuse with ~~Prim's~~ (S.4)

Perform the Nearest-Neighbour algorithm on the table of least distances. (You will be told the starting point).

easier than 'shortcuts' for large networks

(S.2) ← for the practical problem

Find an MST using Prim's or Kruskal.

Double the weight of the MST to get an initial upper bound.

Seek shortcuts (by inspection) to get your upper bound to be "good enough" (below some stated point).

The weight of the MST is an upper bound.

Chapter 6 - Linear Programming

* To formulate a problem as a Linear Programming Problem:

1. Define the decision variables (x, y, z, etc.)
2. State the objective function (along with the instruction "maximise" or "minimise")
3. Write the constraints as inequalities.

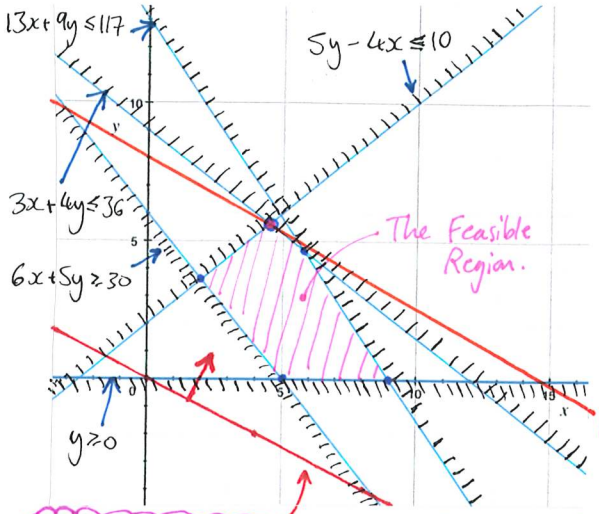
Consider the following problem:

Max. $P = x + 2y$

subject to
$$3x + 4y \leq 36$$
$$13x + 9y \leq 117$$
$$5y - 4x \leq 10$$
$$6x + 5y \geq 30$$
$$y \geq 0$$

Method 1 (The "Ruler Method")

Having drawn the constraints, use a ruler to find the line of the objective function when $P = 0$, then increase x and y until increasing any more will take you outside the feasible region.



Under method 1, draw this initial line, and extend parallel lines through the feasible region until you reach the edge.

The pink dot is now the optimal solution. Solve for the intersection point through simultaneous equations.

Method 2 (The "Vertex Method")

Find the vertices of the feasible region (by inspection, or by multiple systems of simultaneous equations). Then, evaluate P for each pair of x, y (here there are 5 vertices so 5 pairs of x, y), and choose the largest value of P (or smallest if you're minimising).

Method 3 (The Simplex Algorithm) See below ↓

Chapter 7 - The Simplex Algorithm

START

Constraints have only $\leq$ involved.

In this case create equalities by introducing slack variables

Perform the: NORMAL SIMPLEX

Constraints have at least one $\geq$ involved (as well as $\leq$).

In this case create equalities by introducing ① slack variables for $\leq$ constraints and ② for $\geq$ constraints: subtract one surplus value and add one artificial variable

Perform the: 2-STAGE SIMPLEX

or

Perform the: BIG-M METHOD

NB: Simplex always maximises.
Therefore, to solve "minimise $P = x - y + z$", change the problem to: "maximise $-P = -x + y - z$" and keep all the constraints the same.