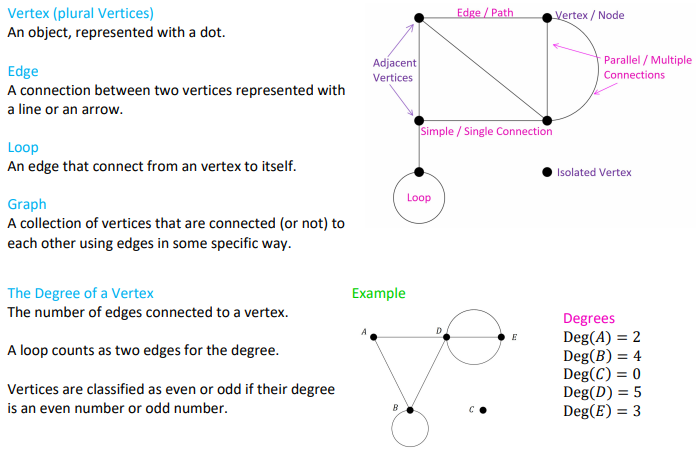
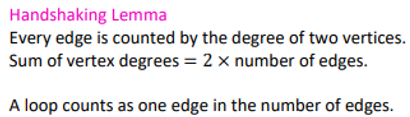
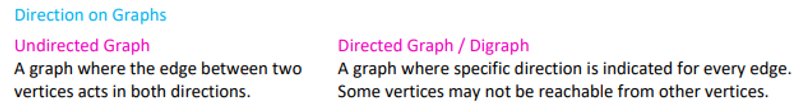
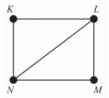
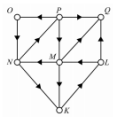
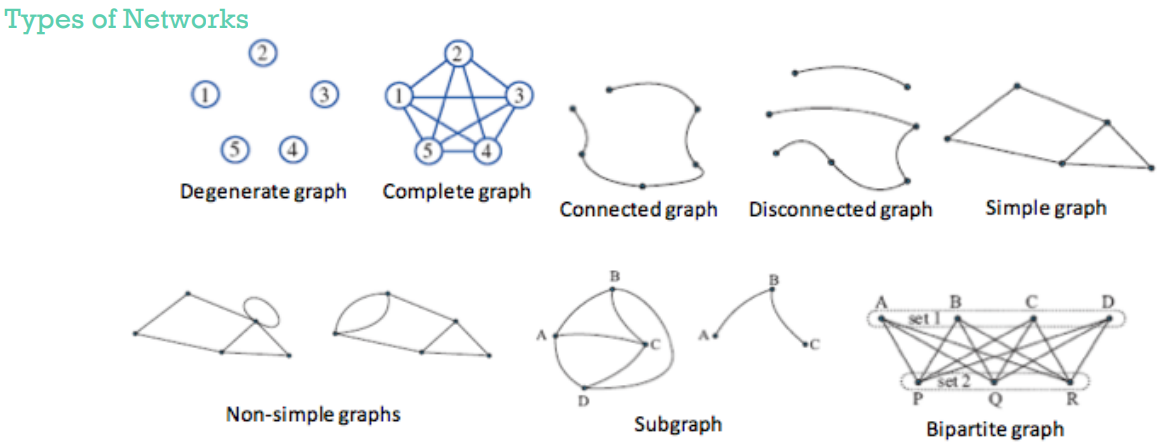
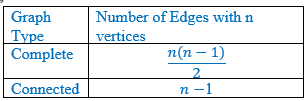
Networks

**** 



**Undirected Graphs**

## ***Types of graphs***

**Simple graph** – No loops or duplicate edges.

**Isolated vertex** – A graph has an isolated vertex if there is a vertex that is not connected to another vertex by an edge.

**Degenerate graph** – Degenerate graphs have all vertices isolated. Therefore, there are no edges in the graph at all.

**Connected graph** – Each vertex is either directly or indirectly connected to every other vertex.

**Bridge** – A bridge is an edge that when removed makes the graph unconnected.

**Subgraph** – Are graphs that are part of larger graphs.

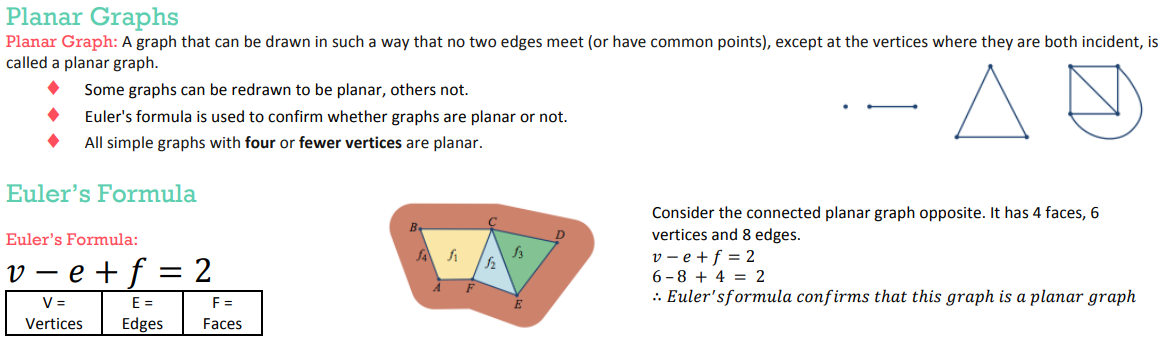
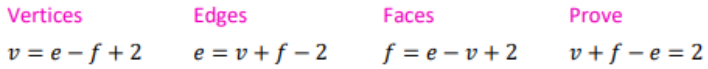
**Equivalent (isomorphic) graph** – Look different but have the same information

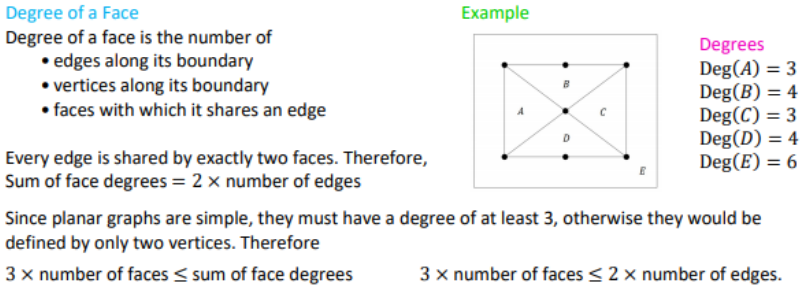
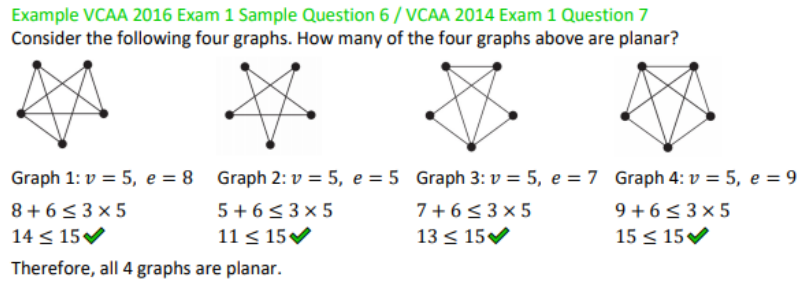
**Complete graph** – Every vertex has a direct connection to every other vertex.

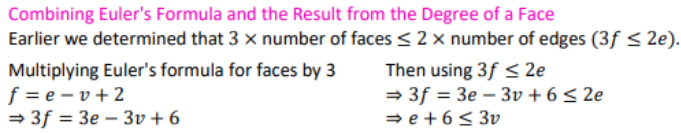
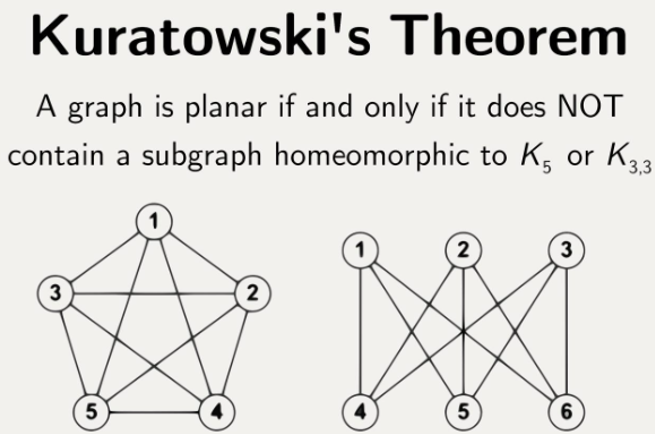
**Bipartite Graph** – A bipartite graph is a graph whose set of vertices can be split into two subsets X and Y in such a way that each edge of the graph joins a vertex in X and a vertex in Y.

**Isomorphic graphs** –Two graphs have: ① same numbers of edges and vertices; ② corresponding vertices have the same degree and the edges connect the same vertices.

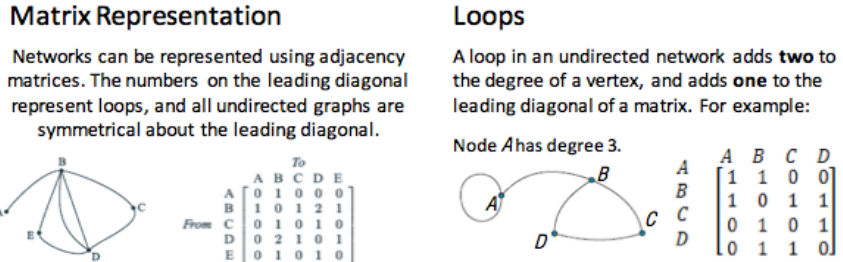
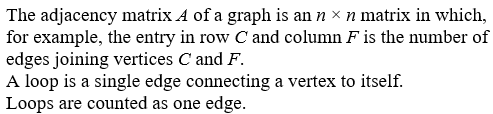
## ***Planar Graphs & Euler’s Formula***

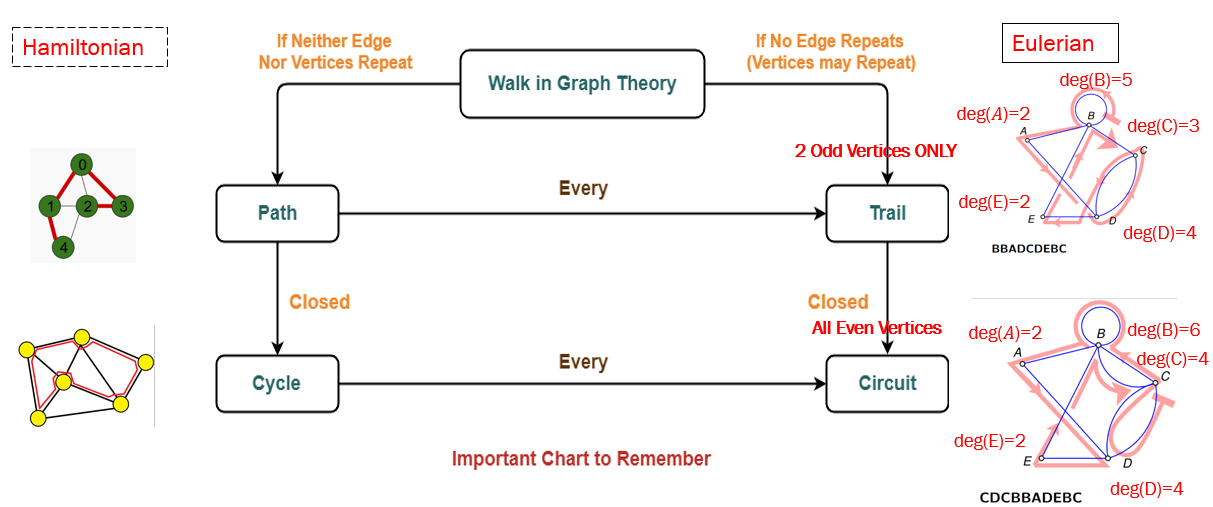
 

## ***Adjacency Matrix Representation***

## ***Euler & Hamilton***



**Travelling in graphs**

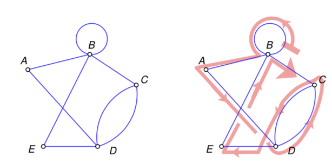
**Route** – A description of your travels, given by the vertices visited in the order they are visited.

**Walk** – A walk can be any type of journey within a graph, you can walk wherever you wish.

**Trail** – A special kind of walk, you **can’t repeat** any of the **edges** that you have taken, but you can revisit vertices.

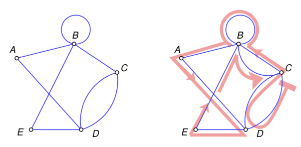
**Path** – A path is a special kind of trail, with a path you **can’t repeat** any **edges or vertices**.

**Eulerian trails and circuits**

**Eulerian trails** – Is a trail in which every **edge** is visited **once**. Vertices can be repeated.

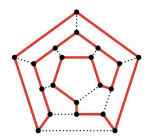
A Eulerian trail will only exist if:

* The graph is connected
* The graph has exactly two vertices of an odd degree

**Eulerian circuit** – Is a Eulerian trail (travels every **edge once**) that begins and ends from the same vertex.

A Eulerian circuit will only exist if:

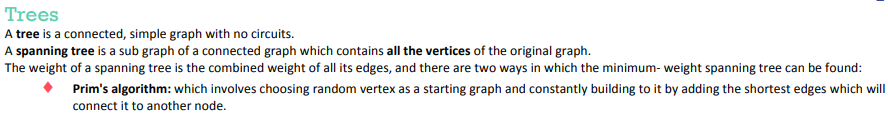
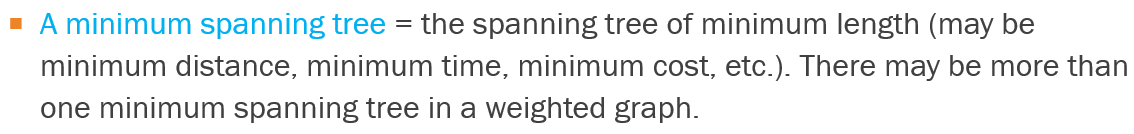
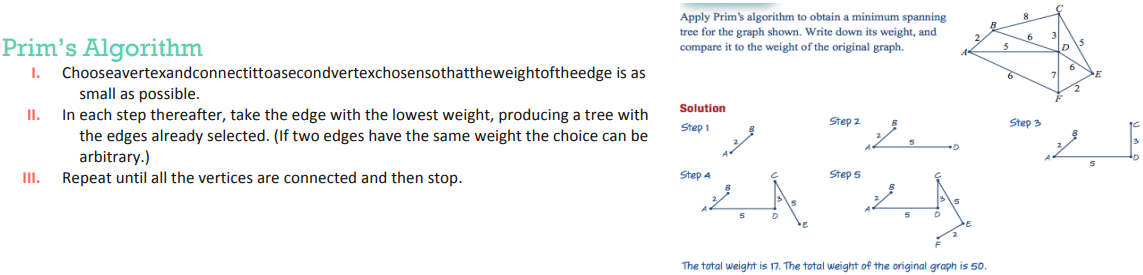
* The graph is connected
* All the vertices have an even degree

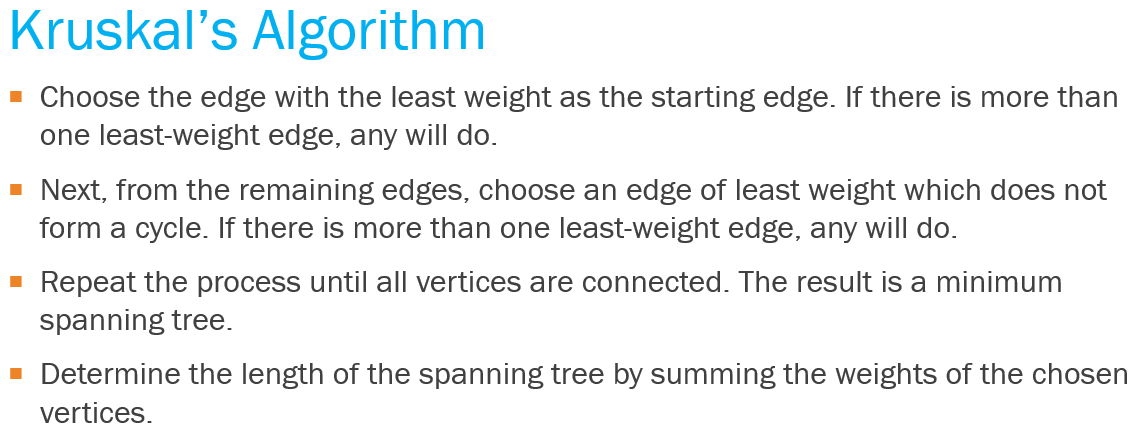
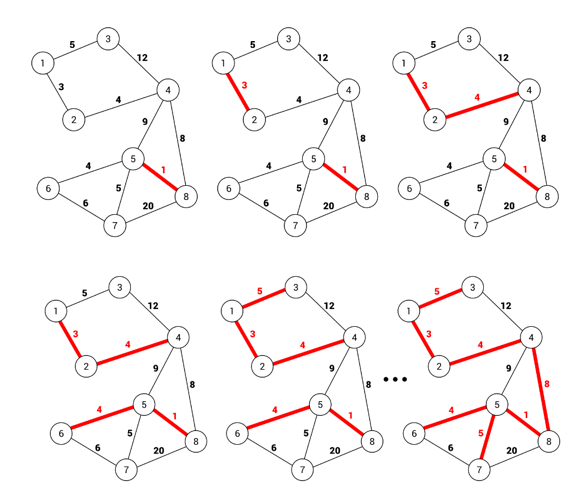
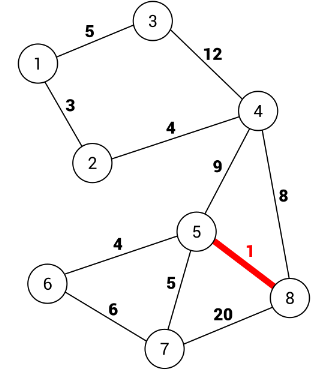
**Hamiltonian paths and cycles**

**Hamiltonian path** – Is a path that visits all of the **vertices** in a graph **only once**.

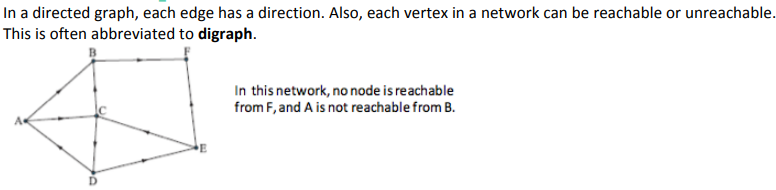
**Hamiltonian cycle** – Is a cycle that visits every vertex and begins and ends at the same vertex.

## ***Weighted Graphs***

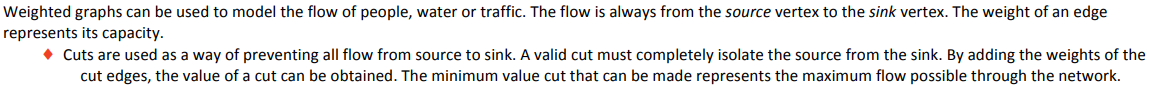
 

**Directed Graphs**



## ***Network Flows***



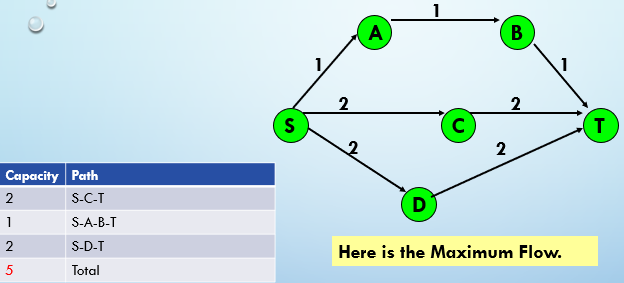
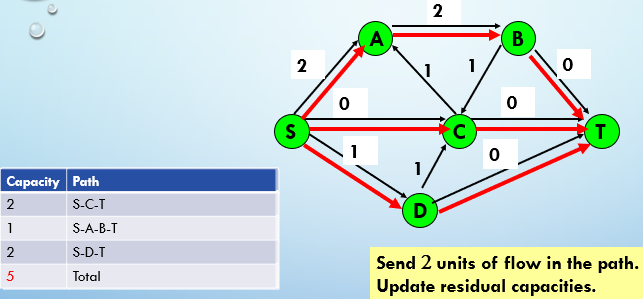
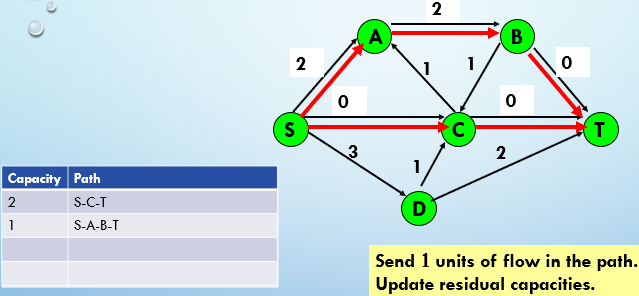
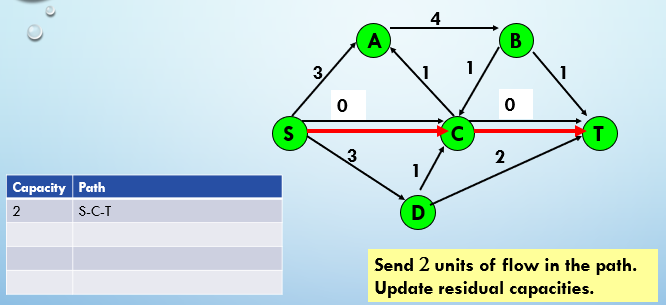
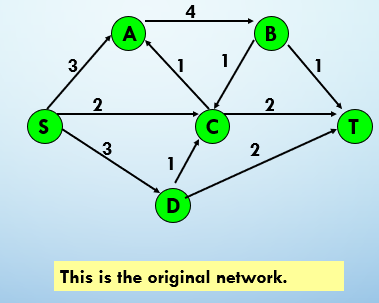
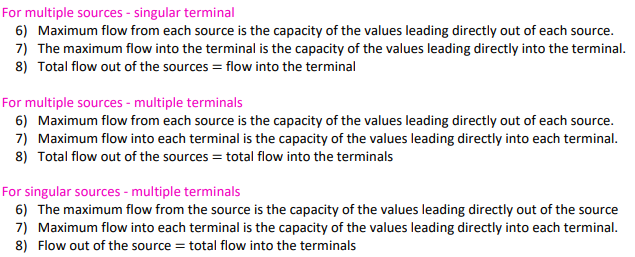
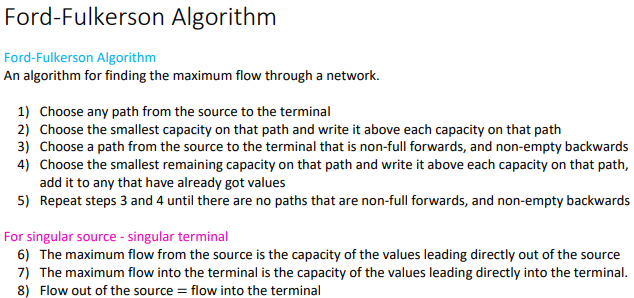
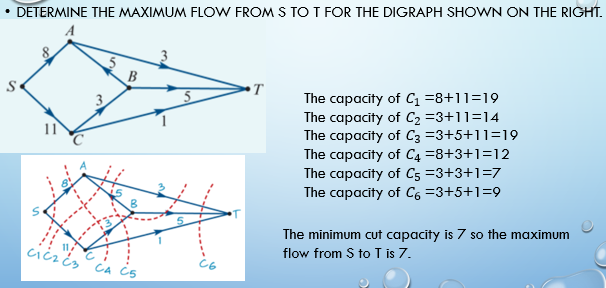
**Capacity**

The capacity of an edge is the maximum amount that can flow through it.

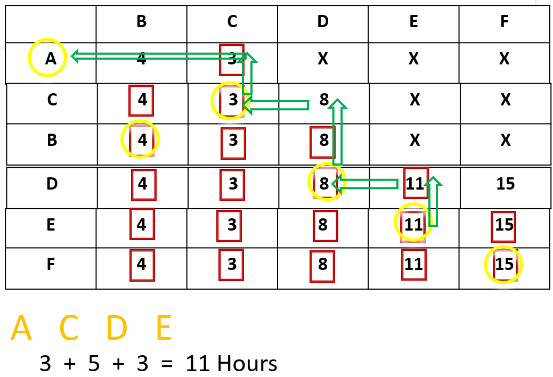
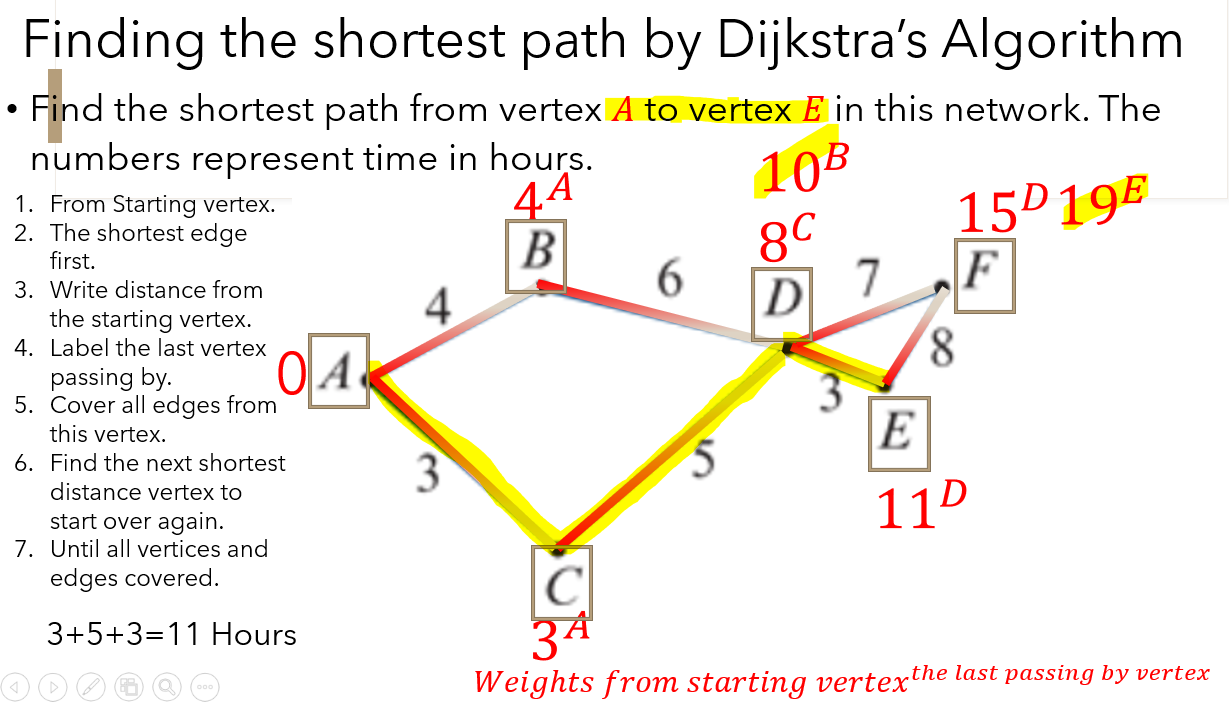
The capacity of a cut is the sum of the weights of the edges in the cut.

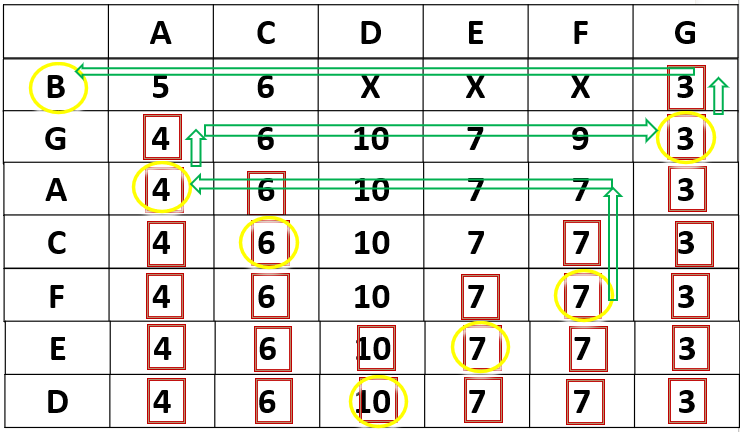
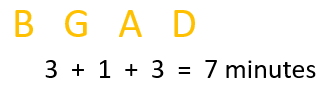
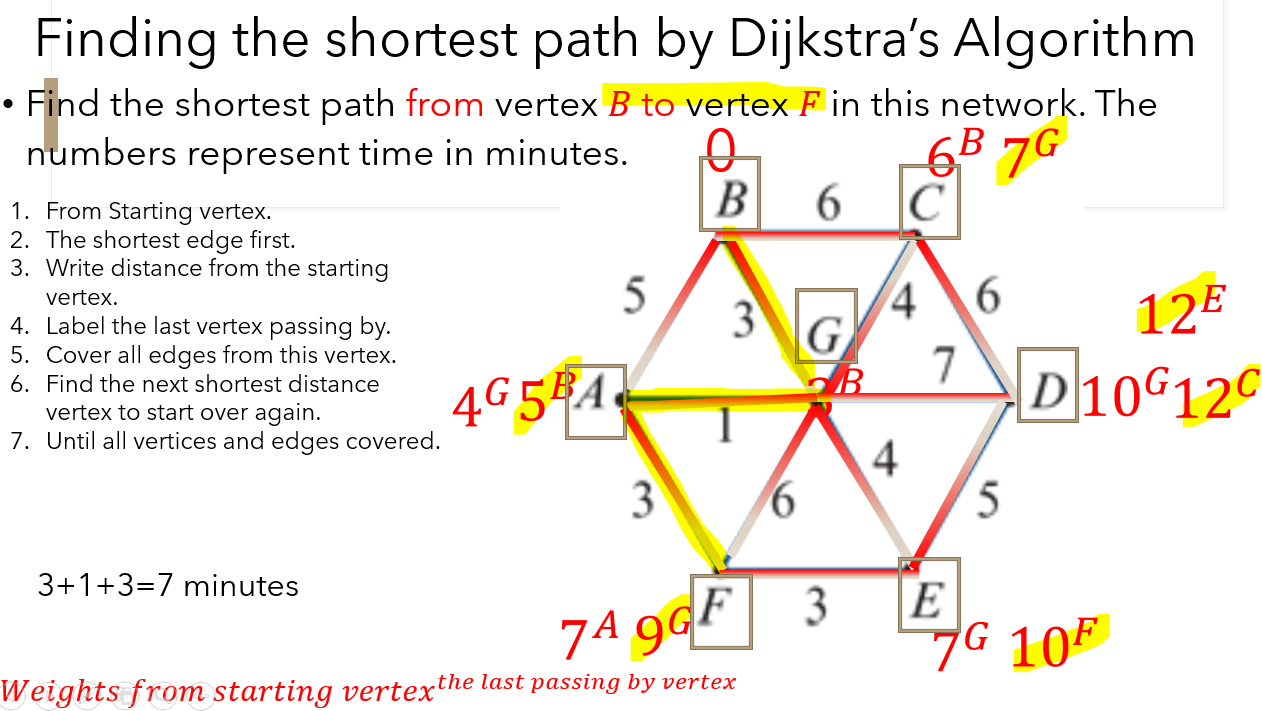
The capacity of a network is the maximum amount that can flow from the sink to the source.

****

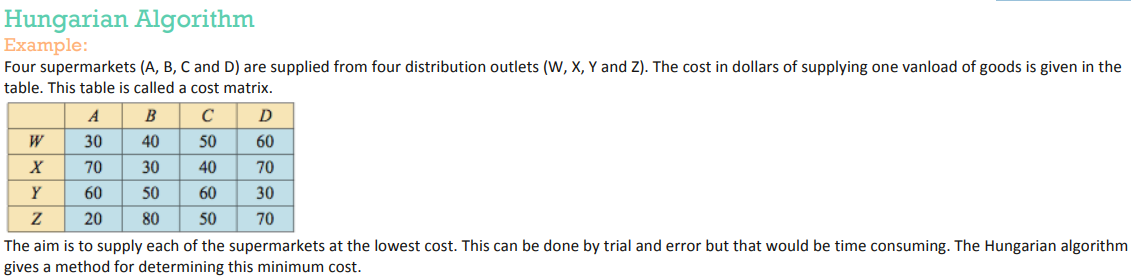
****

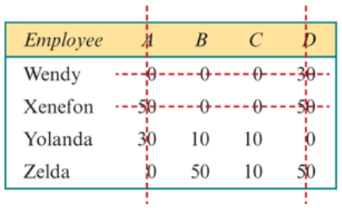
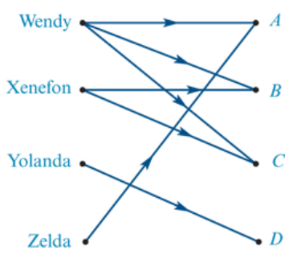
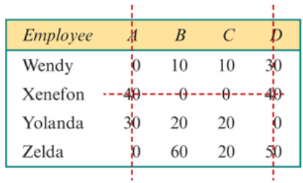
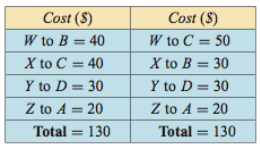
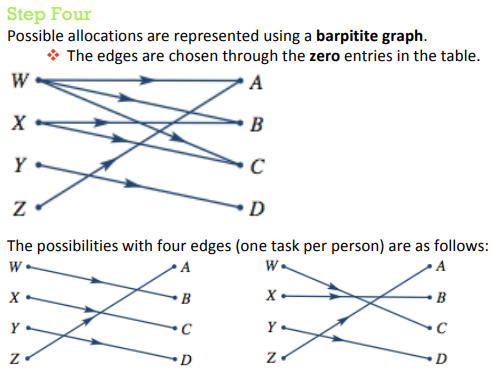
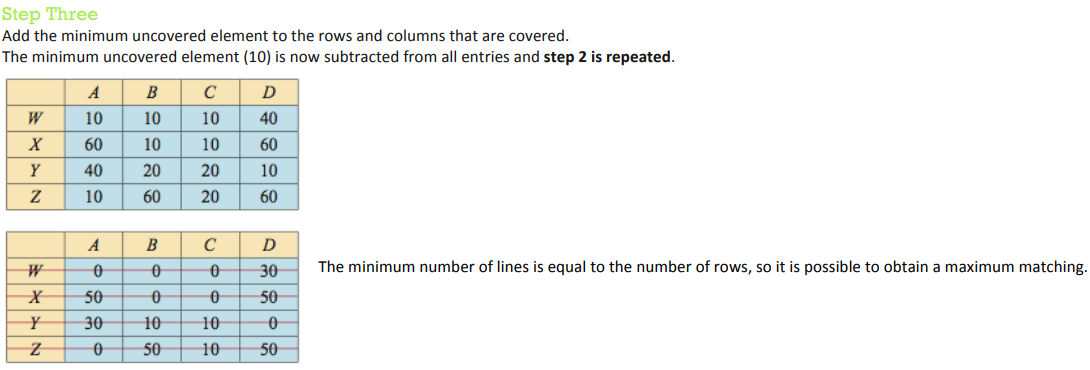
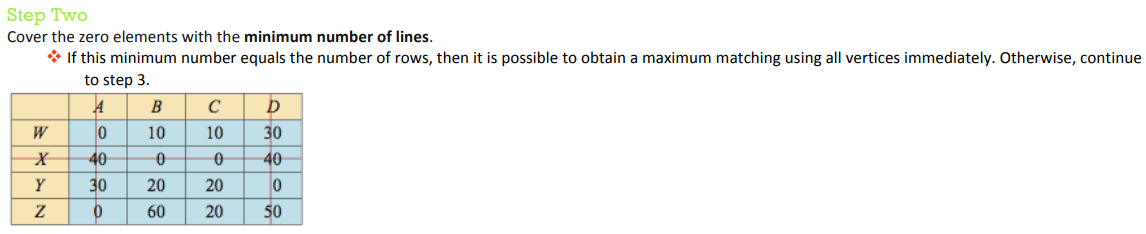
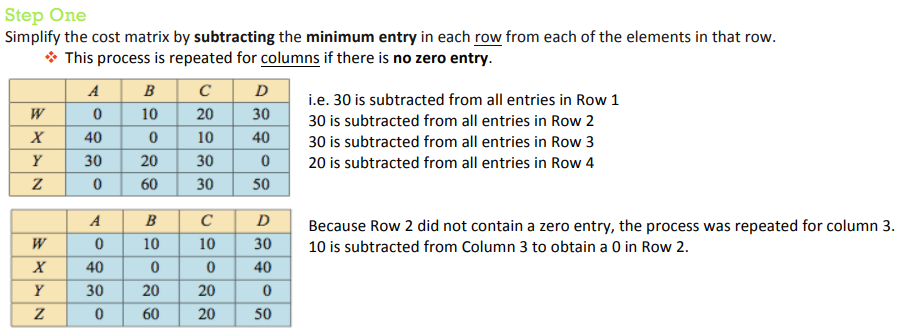
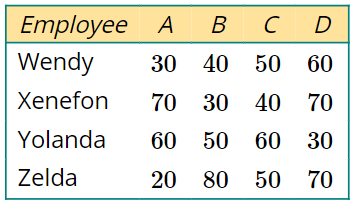
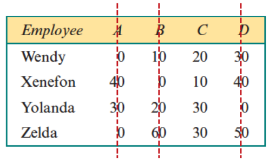
## ***Shortest Path Problem***

** **

** **

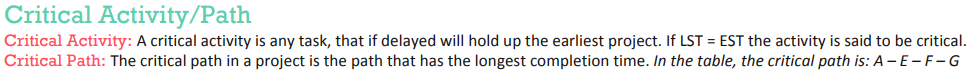
## ***Matching & Allocation Problems***



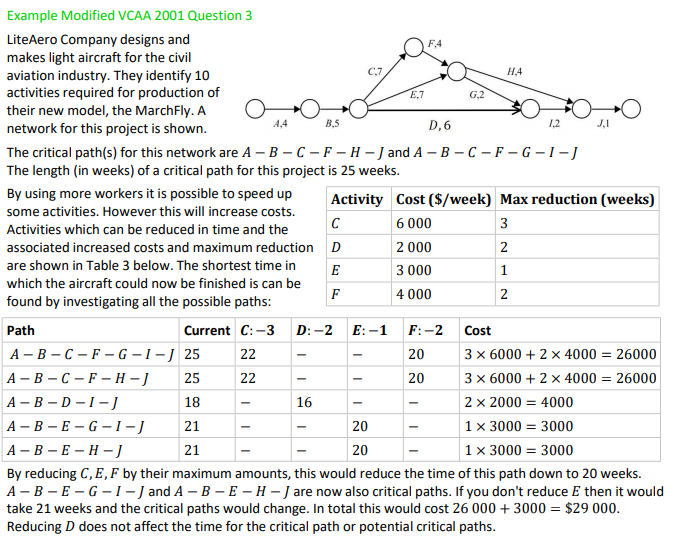
********

## ***Critical Path Problems***

## 



## 



## ***Different Types of Greedy Algorithm***

Prim's Minimal Spanning Tree Algorithm

Kruskal’s Minimal Spanning Tree algorithm

Dijkstra's Shortest Path Algorithm

Ford-Fulkerson Networks Flows Algorithm

## ***Mathematical Terminologies***

|  |  |  |  |
| --- | --- | --- | --- |
| **Undirected Graphs** | | **Directed Graphs** | |
| **Terminologies** | **Algorithm** | **Terminologies** | **Algorithm** |
| Eulerian trails | Exactly 2 vertices of an odd degree | The Maximum Flow 8.1 | Ford-Fulkerson Algorithm |
| Eulerian circuits | All vertices even degree | The Shortest Path | Dijkstra's Algorithm |
| Hamiltonian paths | Visits all of the vertices in a graph only once | Matching & Allocation Problems | Hungarian Algorithm 8.2 |
| Hamiltonian cycles | Visit All vertices, begin & end @ the same vertex | Critical Path Problems | Forward scanning = Biggest Number  Backward scanning = Smallest Number  Float = LST―EST |
| Minimal Spanning Tree | Prim's Algorithm, Kruskal’s Algorithm |

**Hungarian Algorithm Task Match Working Steps**

1. **Row deduction (Min. No. of each row to deduct)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tasks  **Person** | Task 1 | Task 2 | Task 3 | Task 4 | Task 5 |
| **Person 1** |  |  |  |  |  |
| **Person 2** |  |  |  |  |  |
| **Person 3** |  |  |  |  |  |
| **Person 4** |  |  |  |  |  |
| **Person 5** |  |  |  |  |  |

1. **Column deduction (Min. No. of each non 0 column)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tasks  **Person** | Task 1 | Task 2 | Task 3 | Task 4 | Task 5 |
| **Person 1** |  |  |  |  |  |
| **Person 2** |  |  |  |  |  |
| **Person 3** |  |  |  |  |  |
| **Person 4** |  |  |  |  |  |
| **Person 5** |  |  |  |  |  |

1. Line fitting to cover Max. 0 possible ≥ No. of tasks

≥ More than or equal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tasks  **Person** | Task 1 | Task 2 | Task 3 | Task 4 | Task 5 |
| **Person 1** |  |  |  |  |  |
| **Person 2** |  |  |  |  |  |
| **Person 3** |  |  |  |  |  |
| **Person 4** |  |  |  |  |  |
| **Person 5** |  |  |  |  |  |

↓Enough lines to **step 6** ↓ Not enough lines to **step 4**

1. Intersection addition uncovered deduction (use uncovered Min. No)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tasks  **Person** | Task 1 | Task 2 | Task 3 | Task 4 | Task 5 |
| **Person 1** |  |  |  |  |  |
| **Person 2** |  |  |  |  |  |
| **Person 3** |  |  |  |  |  |
| **Person 4** |  |  |  |  |  |
| **Person 5** |  |  |  |  |  |

[

1. Line fitting to cover Max. 0 possible ≥ No. of tasks

≥ More t**han or equal**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tasks  **Person** | Task 1 | Task 2 | Task 3 | Task 4 | Task 5 |
| **Person 1** |  |  |  |  |  |
| **Person 2** |  |  |  |  |  |
| **Person 3** |  |  |  |  |  |
| **Person 4** |  |  |  |  |  |
| **Person 5** |  |  |  |  |  |

1. Bipartite task graph matching (0=line match)

|  |  |  |
| --- | --- | --- |
| Person 1 |  | Task 1 |
| Person 2 |  | Task 2 |
| Person 3 |  | Task 3 |
| Person 4 |  | Task 4 |
| Person 5 |  | Task 5 |

1. Task Allocation (possible 2 solutions)

|  |  |  |  |
| --- | --- | --- | --- |
| Person 1 |  | Person 1 |  |
| Person 2 |  | Person 2 |  |
| Person 3 |  | Person 3 |  |
| Person 4 |  | Person 4 |  |
| Person 5 |  | Person 5 |  |

**Hungarian Algorithm Task Match Working Steps**

1. **Row deduction (Min. No. of each row to deduct)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Person  **Task** | **A** | **B** | **C** | **D** | **E** |
| **1** |  |  |  |  |  |
| **2** |  |  |  |  |  |
| **3** |  |  |  |  |  |
| **4** |  |  |  |  |  |
| **5** |  |  |  |  |  |

1. **Column deduction (Min. No. of each non 0 column)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Person  **Task** | **A** | **B** | **C** | **D** | **E** |
| **1** |  |  |  |  |  |
| **2** |  |  |  |  |  |
| **3** |  |  |  |  |  |
| **4** |  |  |  |  |  |
| **5** |  |  |  |  |  |

1. Line fitting to cover Max. 0 possible ≥ No. of tasks

≥ More than or equal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Person  **Task** | **A** | **B** | **C** | **D** | **E** |
| **1** |  |  |  |  |  |
| **2** |  |  |  |  |  |
| **3** |  |  |  |  |  |
| **4** |  |  |  |  |  |
| **5** |  |  |  |  |  |

↓Enough lines to **step 6** ↓ Not enough lines to **step 4**

1. Intersection addition uncovered deduction (use uncovered Min. No)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Person  **Task** | **A** | **B** | **C** | **D** | **E** |
| **1** |  |  |  |  |  |
| **2** |  |  |  |  |  |
| **3** |  |  |  |  |  |
| **4** |  |  |  |  |  |
| **5** |  |  |  |  |  |

1. Line fitting to cover Max. 0 possible ≥ No. of tasks

≥ More than or equal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Person  **Task** | **A** | **B** | **C** | **D** | **E** |
| **1** |  |  |  |  |  |
| **2** |  |  |  |  |  |
| **3** |  |  |  |  |  |
| **4** |  |  |  |  |  |
| **5** |  |  |  |  |  |

1. Bipartite task graph matching (0=line match)

|  |  |  |
| --- | --- | --- |
| **1** |  | **A** |
| **2** |  | **B** |
| **3** |  | **C** |
| **4** |  | **D** |
| **5** |  | **E** |

1. Task Allocation (possible 2 solutions)

|  |  |  |  |
| --- | --- | --- | --- |
| **1** |  | **1** |  |
| **2** |  | **2** |  |
| **3** |  | **3** |  |
| **4** |  | **4** |  |
| **5** |  | **5** |  |

**8.1 Using “NetworkFlow” TemplateA screenshot of a computer

Description automatically generated A white background with black text

Description automatically generated**

Step 1 Sample graph: **shift** **lette**r for new vertexA screenshot of a computer

Description automatically generated

Step 2 Press **Enter** to add Max Flow=3+4+1=8 in Green

**A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated**

Step 3 Min Cut is showing as following in Orange

**A screenshot of a computer

Description automatically generated**

**8.2 Using “Hungarian Algorithms” Template**

Step 1**:** Define Matrix in file 1.3**A screenshot of a table

Description automatically generated**A screenshot of a computer

Description automatically generated

Step 2: Go back to file 1.2 and press **MENU🡪 Matrix🡪 User-defined**

**A screenshot of a computer

Description automatically generated**

Step 3: Enter the defined matrix name here

**A screenshot of a computer

Description automatically generated**

Step 4: Once you see all matrix number, press **ENTER**

**A screenshot of a computer

Description automatically generated**

Step 5: Compare the result with manual calculation

**A screenshot of a table

Description automatically generatedA black text with black letters

Description automatically generated**