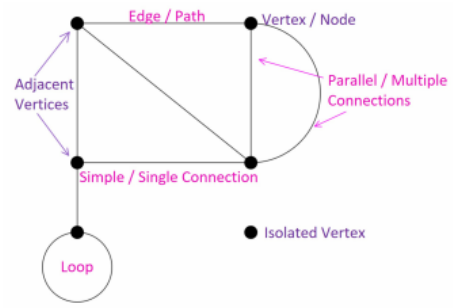
Networks

Vertex (plural Vertices)
An object, represented with a dot.

Edge
A connection between two vertices represented with a line or an arrow.

Loop
An edge that connect from an vertex to itself.

Graph
A collection of vertices that are connected (or not) to each other using edges in some specific way.

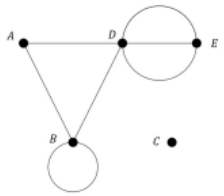


The Degree of a Vertex
The number of edges connected to a vertex.

A loop counts as two edges for the degree.

Vertices are classified as even or odd if their degree is an even number or odd number.

Example



Degrees

$\text{Deg}(A) = 2$
$\text{Deg}(B) = 4$
$\text{Deg}(C) = 0$
$\text{Deg}(D) = 5$
$\text{Deg}(E) = 3$

Handshaking Lemma
Every edge is counted by the degree of two vertices.
Sum of vertex degrees = 2 × number of edges.

A loop counts as one edge in the number of edges.

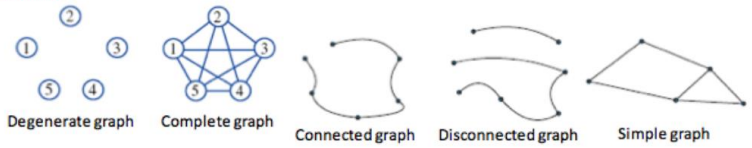
Direction on Graphs

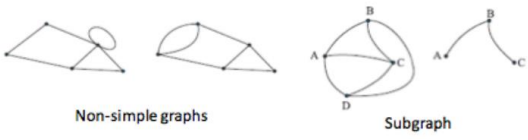
Undirected Graph
A graph where the edge between two vertices acts in both directions.

Directed Graph / Digraph
A graph where specific direction is indicated for every edge.
Some vertices may not be reachable from other vertices.

Undirected Graphs

Types of Networks





Graph Type	Number of Edges with n vertices
Complete	$\frac{n(n-1)}{2}$
Connected	$n-1$

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

Planar Graphs

Planar Graph: A graph that can be drawn in such a way that no two edges meet (or have common points), except at the vertices where they are both incident, is called a planar graph.

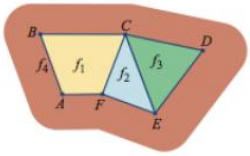
- Some graphs can be redrawn to be planar, others not.
- Euler's formula is used to confirm whether graphs are planar or not.
- All simple graphs with **four** or **fewer vertices** are planar.

Euler's Formula

Euler's Formula:

$$v - e + f = 2$$

V = Vertices	E = Edges	F = Faces



Consider the connected planar graph opposite. It has 4 faces, 6 vertices and 8 edges.

$v - e + f = 2$

$6 - 8 + 4 = 2$

$\therefore$ *Euler's formula confirms that this graph is a planar graph*

Vertices	Edges	Faces	Prove
$v = e - f + 2$	$e = v + f - 2$	$f = e - v + 2$	$v + f - e = 2$

Euler's Formula/Degree of a Face

$e + 6 \leq 3v$

Degree of a Face

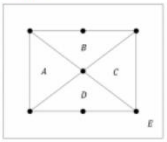
Degree of a face is the number of
- edges along its boundary
- vertices along its boundary
- faces with which it shares an edge

Every edge is shared by exactly two faces. Therefore,
Sum of face degrees = 2 × number of edges

Since planar graphs are simple, they must have a degree of at least 3, otherwise they would be defined by only two vertices. Therefore

3 × number of faces ≤ sum of face degrees 3 × number of faces ≤ 2 × number of edges.

Example



Degrees

$\text{Deg}(A) = 3$
$\text{Deg}(B) = 4$
$\text{Deg}(C) = 3$
$\text{Deg}(D) = 4$
$\text{Deg}(E) = 6$

Example VCAA 2016 Exam 1 Sample Question 6 / VCAA 2014 Exam 1 Question 7

Consider the following four graphs. How many of the four graphs above are planar?




Graph 1: $v = 5,\ e = 8$	Graph 2: $v = 5,\ e = 5$	Graph 3: $v = 5,\ e = 7$	Graph 4: $v = 5,\ e = 9$
$8 + 6 \leq 3 \times 5$	$5 + 6 \leq 3 \times 5$	$7 + 6 \leq 3 \times 5$	$9 + 6 \leq 3 \times 5$
$14 \leq 15$ ✔	$11 \leq 15$ ✔	$13 \leq 15$ ✔	$15 \leq 15$ ✔

Therefore, all 4 graphs are planar.

Combining Euler's Formula and the Result from the Degree of a Face

Earlier we determined that 3 × number of faces ≤ 2 × number of edges ($3f \leq 2e$).

Multiplying Euler's formula for faces by 3
$f = e - v + 2$
$\Rightarrow 3f = 3e - 3v + 6$

Then using $3f \leq 2e$
$\Rightarrow 3f = 3e - 3v + 6 \leq 2e$
$\Rightarrow e + 6 \leq 3v$

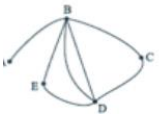
Kuratowski's Theorem

A graph is planar if and only if it does NOT contain a subgraph homeomorphic to K_5 or $K_{3,3}$

Adjacency Matrix Representation

Matrix Representation

Networks can be represented using adjacency matrices. The numbers on the leading diagonal represent loops, and all undirected graphs are symmetrical about the leading diagonal.

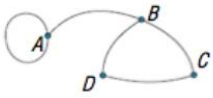


From \ To	A	B	C	D	E
A	0	1	0	0	0
B	1	0	1	2	1
C	0	1	0	1	0
D	0	2	1	0	1
E	0	1	0	1	0

Loops

A loop in an undirected network adds **two** to the degree of a vertex, and adds **one** to the leading diagonal of a matrix. For example:

Node *A* has degree 3.

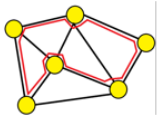


	A	B	C	D
A	1	1	0	0
B	1	0	1	1
C	0	1	0	1
D	0	1	1	0

The adjacency matrix A of a graph is an $n \times n$ matrix in which, for example, the entry in row C and column F is the number of edges joining vertices C and F.
A loop is a single edge connecting a vertex to itself.
Loops are counted as one edge.

Euler & Hamilton

Hamiltonian



Walk in Graph Theory

If Neither Edge Nor Vertices Repeat → **Path**

If No Edge Repeats (Vertices may Repeat) → **Trail**

Path — Every → **Trail**

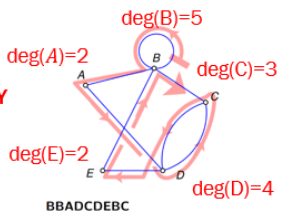
Path — Closed → **Cycle**

Trail — Closed (All Even Vertices) → **Circuit**

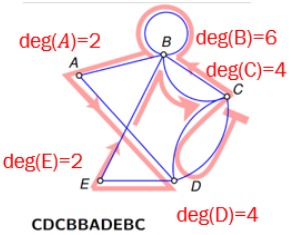
Cycle — Every → **Circuit**

Important Chart to Remember

Eulerian



BBADCDEBC



CDCBBADEBC

Travelling in graphs

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

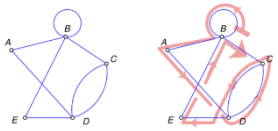
Walk – A walk can be any type of journey within a graph, you can walk wherever you wish.

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

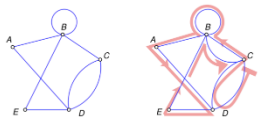
Eulerian trails – Is a trail in which every **edge** is visited **once**. Vertices can be repeated.



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.



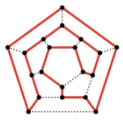
A Eulerian circuit will only exist if:

- 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

Trees

A **tree** is a connected, simple graph with no circuits.

A **spanning tree** is a sub graph of a connected graph which contains **all the vertices** of the original graph.

The weight of a spanning tree is the combined weight of all its edges, and there are two ways in which the minimum- weight spanning tree can be found:

- **Prim's algorithm:** which involves choosing random vertex as a starting graph and constantly building to it by adding the shortest edges which will connect it to another node.

Prim's Algorithm

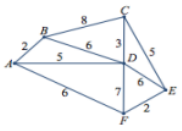
I. Chooseavertexandconnectittoasecondvertexchosensothattheweightoftheedge is as small as possible.

II. In each step thereafter, take the edge with the lowest weight, producing a tree with the edges already selected. (If two edges have the same weight the choice can be arbitrary.)

III. Repeat until all the vertices are connected and then stop.

- A minimum spanning tree = the spanning tree of minimum length (may be minimum distance, minimum time, minimum cost, etc.). There may be more than one minimum spanning tree in a weighted graph.

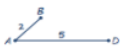
Apply Prim's algorithm to obtain a minimum spanning tree for the graph shown. Write down its weight, and compare it to the weight of the original graph.



Solution

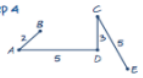
Step 1

Step 2

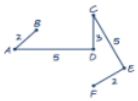


Step 3

Step 4



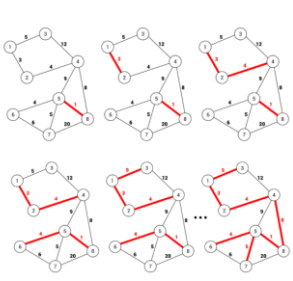
Step 5

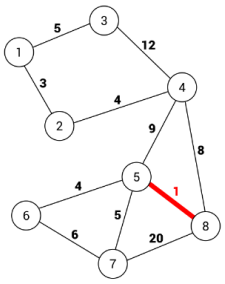


The total weight is 17. The total weight of the original graph is 50.

Kruskal's Algorithm

- Choose the edge with the least weight as the starting edge. If there is more than one least-weight edge, any will do.
- Next, from the remaining edges, choose an edge of least weight which does not form a cycle. If there is more than one least-weight edge, any will do.
- Repeat the process until all vertices are connected. The result is a minimum spanning tree.
- Determine the length of the spanning tree by summing the weights of the chosen vertices.

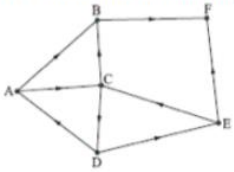




Directed Graphs

In a directed graph, each edge has a direction. Also, each vertex in a network can be reachable or unreachable.

This is often abbreviated to **digraph**.



In this network, no node is reachable from F, and A is not reachable from B.

Network Flows

Weighted graphs can be used to model the flow of people, water or traffic. The flow is always from the *source* vertex to the *sink* vertex. The weight of an edge represents its capacity.

- Cuts are used as a way of preventing all flow from source to sink. A valid cut must completely isolate the source from the sink. By adding the weights of the cut edges, the value of a cut can be obtained. The minimum value cut that can be made represents the maximum flow possible through the network.

Capacity

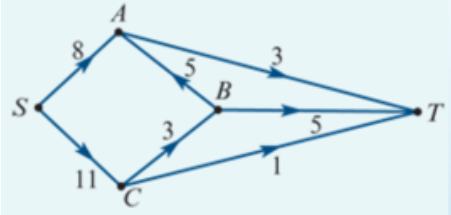
The capacity of an edge is the maximum amount that can flow through it.

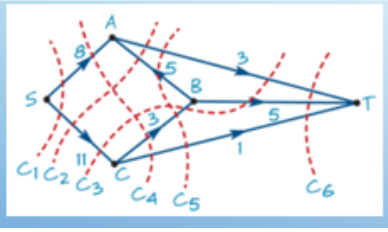
The capacity of a cut is the sum of the weights of the edges in the cut.

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

• THE MINIMUM CUT CAPACITY = THE MAXIMUM FLOW

• DETERMINE THE MAXIMUM FLOW FROM S TO T FOR THE DIGRAPH SHOWN ON THE RIGHT.





The capacity of C_1 =8+11=19
The capacity of C_2 =3+11=14
The capacity of C_3 =3+5+11=19
The capacity of C_4 =8+3+1=12
The capacity of C_5 =3+3+1=7
The capacity of C_6 =3+5+1=9

The minimum cut capacity is 7 so the maximum flow from S to T is 7.

Ford-Fulkerson Algorithm

Ford-Fulkerson Algorithm

An algorithm for finding the maximum flow through a network.

1) Choose any path from the source to the terminal
2) Choose the smallest capacity on that path and write it above each capacity on that path
3) Choose a path from the source to the terminal that is non-full forwards, and non-empty backwards
4) Choose the smallest remaining capacity on that path and write it above each capacity on that path, add it to any that have already got values
5) Repeat steps 3 and 4 until there are no paths that are non-full forwards, and non-empty backwards

For singular source - singular terminal

6) The maximum flow from the source is the capacity of the values leading directly out of the source
7) The maximum flow into the terminal is the capacity of the values leading directly into the terminal.
8) Flow out of the source = flow into the terminal

For multiple sources - singular terminal

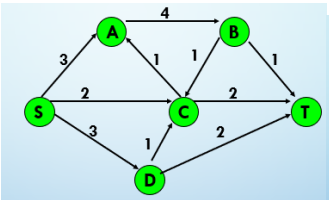
6) Maximum flow from each source is the capacity of the values leading directly out of each source.
7) The maximum flow into the terminal is the capacity of the values leading directly into the terminal.
8) Total flow out of the sources = flow into the terminal

For multiple sources - multiple terminals

6) Maximum flow from each source is the capacity of the values leading directly out of each source.
7) Maximum flow into each terminal is the capacity of the values leading directly into each terminal.
8) Total flow out of the sources = total flow into the terminals

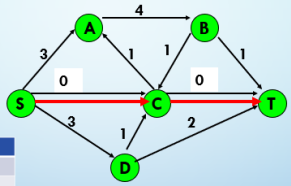
For singular sources - multiple terminals

6) The maximum flow from the source is the capacity of the values leading directly out of the source
7) Maximum flow into each terminal is the capacity of the values leading directly into each terminal.
8) Flow out of the source = total flow into the terminals



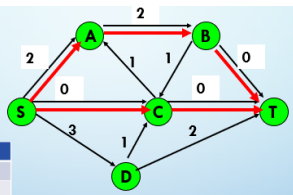
This is the original network.

Capacity	Path
2	S-C-T



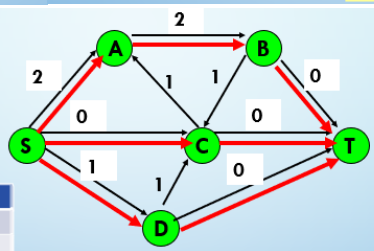
Send 2 units of flow in the path.
Update residual capacities.

Capacity	Path
2	S-C-T
1	S-A-B-T



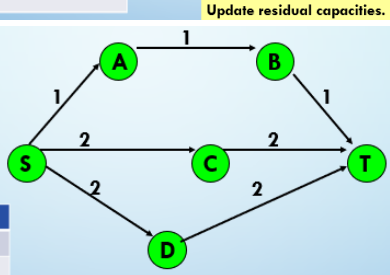
Send 1 units of flow in the path.
Update residual capacities.

Capacity	Path
2	S-C-T
1	S-A-B-T
2	S-D-T
5	Total



Send 2 units of flow in the path.
Update residual capacities.

Capacity	Path
2	S-C-T
1	S-A-B-T
2	S-D-T
5	Total

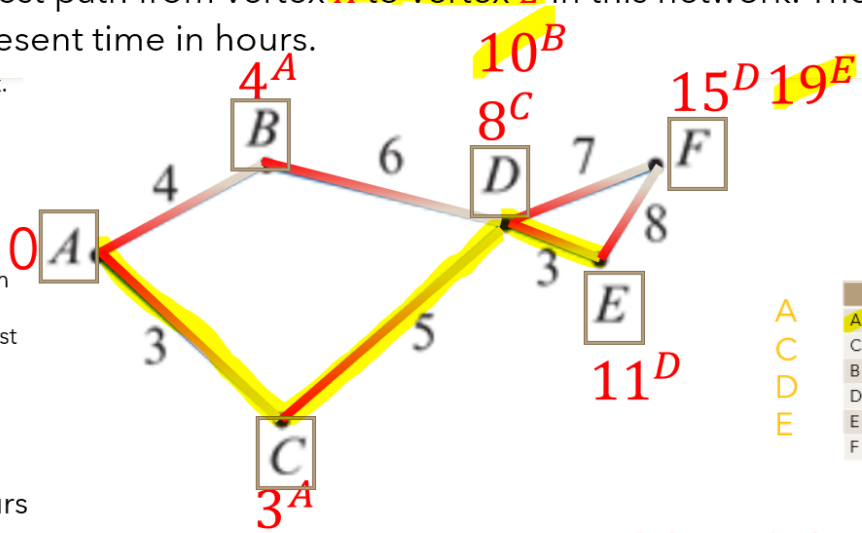


Here is the Maximum Flow.

Finding the shortest path by Dijkstra's Algorithm

- Find the shortest path from vertex A to vertex E in this network. The numbers represent time in hours.

1. From Starting vertex.
2. The shortest edge first.
3. Write distance from the starting vertex.
4. Label the last vertex passing by.
5. Cover all edges from this vertex.
6. Find the next shortest distance vertex to start over again.
7. Until all vertices and edges covered.



A C D E

	B	C	D	E	F
A	4	3			
C		3	8		
B	4				
D			8	11	15
E				11	
F					15

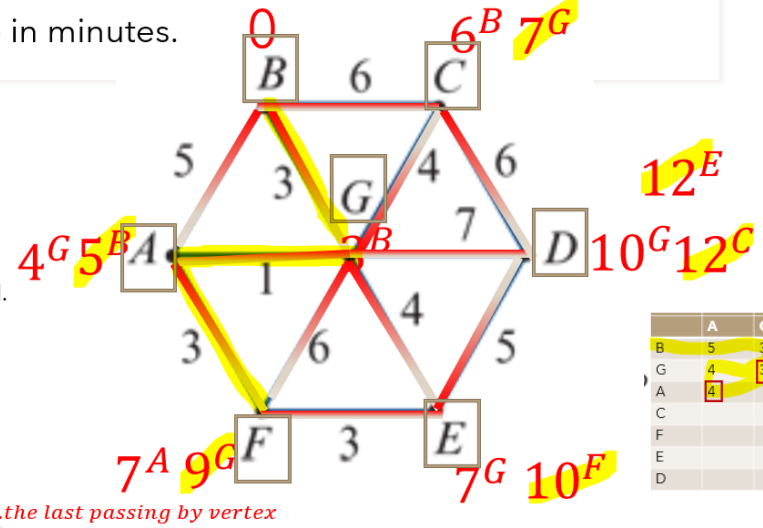
3+5+3=11 Hours

$Weights\ from\ starting\ vertex^{the\ last\ passing\ by\ vertex}$

Finding the shortest path by Dijkstra's Algorithm

- Find the shortest path from vertex B to vertex F in this network. The numbers represent time in minutes.

1. From Starting vertex.
2. The shortest edge first.
3. Write distance from the starting vertex.
4. Label the last vertex passing by.
5. Cover all edges from this vertex.
6. Find the next shortest distance vertex to start over again.
7. Until all vertices and edges covered.



	A	G	C	D	E	F
B	5	3	6			
G	4	3		10	7	
A	4					7
C			6	10		
F						7
E					7	
D				10		

3+1+3=7 minutes

$Weights\ from\ starting\ vertex^{the\ last\ passing\ by\ vertex}$

Matching & Allocation Problems

Hungarian Algorithm

Example:

Four supermarkets (A, B, C and D) are supplied from four distribution outlets (W, X, Y and Z). The cost in dollars of supplying one vanload of goods is given in the table. This table is called a cost matrix.

	A	B	C	D
W	30	40	50	60
X	70	30	40	70
Y	60	50	60	30
Z	20	80	50	70

The aim is to supply each of the supermarkets at the lowest cost. This can be done by trial and error but that would be time consuming. The Hungarian algorithm gives a method for determining this minimum cost.

Step One

Simplify the cost matrix by **subtracting** the **minimum entry** in each row from each of the elements in that row.

- This process is repeated for columns if there is **no zero entry**.

	A	B	C	D
W	0	10	20	30
X	40	0	10	40
Y	30	20	30	0
Z	0	60	30	50

i.e. 30 is subtracted from all entries in Row 1
30 is subtracted from all entries in Row 2
30 is subtracted from all entries in Row 3
20 is subtracted from all entries in Row 4

Employee	A	B	C	D	
Wendy	30	40	50	60	−30
Xenefon	70	30	40	70	−30
Yolanda	60	50	60	30	−30
Zelda	20	80	50	70	−20

Row Minus

	A	B	C	D
W	0	10	10	30
X	40	0	0	40
Y	30	20	20	0
Z	0	60	20	50

Because Row 2 did not contain a zero entry, the process was repeated for column 3.
10 is subtracted from Column 3 to obtain a 0 in Row 2.

Column Minus

Employee	A	B	C	D
Wendy	0	10	20	30
Xenefon	40	0	10	40
Yolanda	30	20	30	0
Zelda	0	60	30	50
			−10	

Step Two

Cover the zero elements with the **minimum number of lines**.

- If this minimum number equals the number of rows, then it is possible to obtain a maximum matching using all vertices immediately. Otherwise, continue to step 3.

	A	B	C	D
W	0	10	10	30
X	40	0	0	40
Y	30	20	20	0
Z	0	60	20	50

	Employee	A	B	C	D	
				−10		
+10	Wendy	0	10	10	30	
+10	Xenefon	40	0	0	40	+10
	Yolanda	30	20	20	0	
	Zelda	0	60	20	50	
				−10		

Intercept Plus & Uncovered Minus

Step Three

Add the minimum uncovered element to the rows and columns that are covered.
The minimum uncovered element (10) is now subtracted from all entries and **step 2 is repeated**.

	A	B	C	D
W	10	10	10	40
X	60	10	10	60
Y	40	20	20	10
Z	10	60	20	60

Employee	A	B	C	D
Wendy	0	0	0	30
Xenefon	50	0	0	50
Yolanda	30	10	10	0
Zelda	0	50	10	50

4 people = Minimum 4 lines

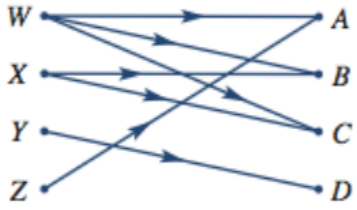
	A	B	C	D
W	0	0	0	30
X	50	0	0	50
Y	30	10	10	0
Z	0	50	10	50

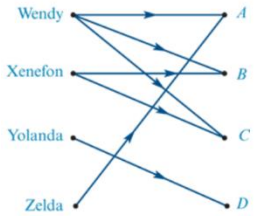
The minimum number of lines is equal to the number of rows, so it is possible to obtain a maximum matching.

Step Four

Possible allocations are represented using a **barpitite graph**.

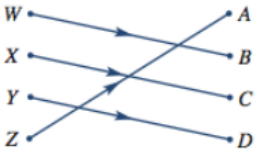
- The edges are chosen through the **zero** entries in the table.





0 in table = A line of Connection

The possibilities with four edges (one task per person) are as follows:



Cost ($)	Cost ($)
W to $B = 40$	W to $C = 50$
X to $C = 40$	X to $B = 30$
Y to $D = 30$	Y to $D = 30$
Z to $A = 20$	Z to $A = 20$
Total $= 130$	**Total** $= 130$

Critical Path Problems

EST	LST

1. Draw a box for each going forward edge/activities.
2. EST = Going forward with **Biggest** Number.
3. LST= Going backward with **Smallest** Number.
4. Float time for each activity = LST — EST @ start of the edge
5. Label / Highlight Critical Path
6. Write minimum completion time.

Conventions of Critical Path Problems

Developing and manufacturing a product frequently involves many interrelated activities. It is often the case that some of these activities cannot be started until other activities are completed.

Two important facts about critical paths are:

I. The weight of the critical path is the minimal length of time required to complete the project.
II. Increasing the time required for any critical activity will also increase the time necessary to complete the project.

Digraphs can be used to represent such situations with the following conditions applying:

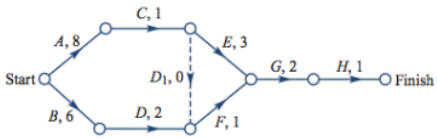
- The edges (or arcs) represent the activities.
- The vertices (or nodes) represent events.
- The start/finish of one or more activities is called an event.
- An edge should lead from a
- A vertex (called the finish node) representing the completion of the project should be included in the network.
- An activity should not be represented by more than one edge in the network.
- Two nodes can be connected directly by, at most, one edge.

Precedence Table

A table that details the events that must occur immediately before an event may begin and its duration.

Activities	Immediate predecessors
A	-
B	-
C	A
D	B
E	C
F	C, D
G	E, F
H	G

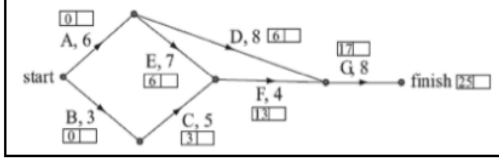
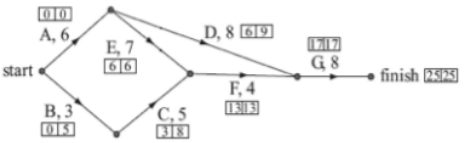
In order to satisfy the final two conventions, it is sometimes necessary to introduce a dummy activity that takes zero time. Following these conventions the weighted digraph can be redrawn.



EST | LST

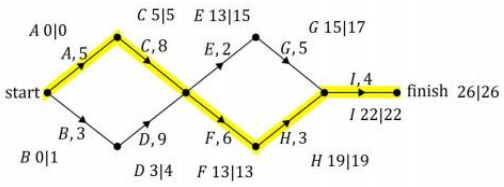
EST	LST

Earliest and Latest Starting Time and Float Time

Earliest Starting Time	Latest Starting Time	Float/Slack Time
The earliest starting time refers to the earliest time the activity can commence. The EST for activities without predecessors is zero. The EST for activities should be the **longest** elapsed time	The latest start time is the latest time an activity can be left if the whole project is to be completed on time. Latest event times are established by working backwards through the network.	For **critical activities** the float time is zero For **non-critical activities** the float is worked out using: $Float\ Time = LST - EST$
		**FLOAT TIMES** A: 0 – 0 = 0 F: 13 – 13 = 0 B: 5 – 0 = 5 G: 17 – 17 = 0 C: 8 – 3 = 5 D: 9 – 6 = 3 E: 6 – 6 = 0

Critical Activity/Path

Critical Activity: A critical activity is any task, that if delayed will hold up the earliest project. If LST = EST the activity is said to be critical.
Critical Path: The critical path in a project is the path that has the longest completion time. *In the table, the critical path is: A – E – F – G*



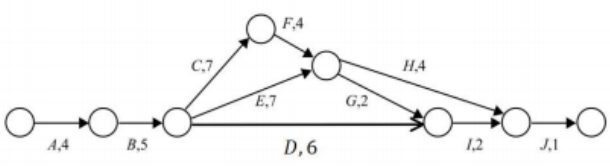
Project Crashing

If a project looks like running overtime, it may be crashed. Crashing involves spending extra money to reduce the time taken by certain activities in order to avoid costs of completing the project late. The following steps illustrate the crashing process:

I. Write down all of the paths from the start node to the finish node, and determine the length of each.
II. Calculate the cost per day of crashing each activity (note that some questions may already have the cost per day, and watch for reductions that must be made in full)
III. Reduce the cheapest (cost per day) activity **on the critical path** by one day.
IV. Calculate the new lengths of all of the paths.
V. Repeat steps three and four, each time using the new longest path after reductions. Stop when the budget is reached or the longest path cannot be reduced. The new critical path is the longest path.

Example Modified VCAA 2001 Question 3

LiteAero Company designs and makes light aircraft for the civil aviation industry. They identify 10 activities required for production of their new model, the MarchFly. A network for this project is shown.



The critical path(s) for this network are $A-B-C-F-H-J$ and $A-B-C-F-G-I-J$
The length (in weeks) of a critical path for this project is 25 weeks.

By using more workers it is possible to speed up some activities. However this will increase costs. Activities which can be reduced in time and the associated increased costs and maximum reduction are shown in Table 3 below. The shortest time in which the aircraft could now be finished is can be found by investigating all the possible paths:

Activity	Cost ($/week)	Max reduction (weeks)
C	6 000	3
D	2 000	2
E	3 000	1
F	4 000	2

Path	Current	$C:-3$	$D:-2$	$E:-1$	$F:-2$	Cost
$A-B-C-F-G-I-J$	25	22	–	–	20	$3\times6000+2\times4000=26000$
$A-B-C-F-H-J$	25	22	–	–	20	$3\times6000+2\times4000=26000$
$A-B-D-I-J$	18	–	16	–	–	$2\times2000=4000$
$A-B-E-G-I-J$	21	–	–	20	–	$1\times3000=3000$
$A-B-E-H-J$	21	–	–	20	–	$1\times3000=3000$

By reducing C, E, F by their maximum amounts, this would reduce the time of this path down to 20 weeks. $A-B-E-G-I-J$ and $A-B-E-H-J$ are now also critical paths. If you don't reduce E then it would take 21 weeks and the critical paths would change. In total this would cost $26\,000+3000=\$29\,000$. Reducing D does not affect the time for the critical path or potential critical paths.

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	Ford-Fulkerson Algorithm
Eulerian circuits	All vertices even degree	The Shortest Path	Dijkstra's Algorithm ④
Hamiltonian paths	Visits all of the vertices in a graph only once ②	Matching & Allocation Problems	Hungarian Algorithm ⑤
Hamiltonian cycles	Visit All vertices, begin & end @ the same vertex ① ⑥	Critical Path Problems	Forward scanning = Biggest Number Backward scanning = Smallest Number Float = LST—EST
Minimal Spanning Tree	Prim's Algorithm, Kruskal's Algorithm ③		